Aplicação de Backup/Restore no Firebird: Crie sua própria ferramenta

Clube Delphi

Ano 13 :: Edição 146 :: R$ 11,90

DevMedia

Minicurso – FastReport
Criação de variáveis, CrossTables e relatórios dinâmicos

MINERAÇÃO DE DADOS

Inteligência da informação com FastScript

LiveBindings na Prática
Entendendo as expressões de binding

Generics e Hash Table
Segurança no tratamento de tipos

ISSN 1517990-7

TENHA ACESSO ILIMITADO A TODO CONTEÚDO DO SITE DEVMEDIA

SEJA UM ASSINANTE MVP!

Tenha acesso garantido a qualquer tema que você possa necessitar!

VIDEOS -AULAS

O assinante MVP acessa todas as vídeo-aulas do site DevMedia. Nenhum outro produto dá essa possibilidade. São mais de **7.000 video-aulas.**

CURSOS

Nosso site possui atualmente **+ de 250 cursos online** com a qualidade que você já conhece e confia. Com o MVP você terá acesso a todos eles.

REVISTAS

Assinante MVP acessa todas as nossas revistas na versão digital, incluindo todas as edições anteriores. Quando pintar aquela dúvida você terá mais de **3.500 artigos para consultar!**

MVP
MOST VALUABLE PROFESSIONAL

A assinatura MVP te dá a oportunidade de aumentar seu conhecimento em diversos assuntos, com qualidade e economia.

Invista na sua carreira se torne um profissional ainda mais completo!

POR APENAS R$59,90 POR MÊS

DevMedia

Acesse: www.devmedia.com.br/mvp

Sumário

Delphi

Delphi — **06 – Mineração de Dados**
[*Giuliano Scombatti Pinto*]

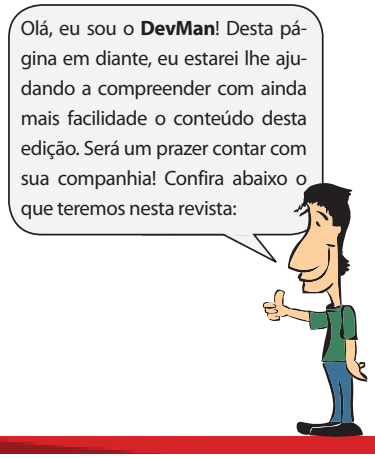
Delphi — **18 – LiveBindings na Prática**
[*Fabrício Hissao Kawata*]

Easy — **26 – Generics e Hash Table**
[*Giuliano Scombatti Pinto*]

Boa Ideia - Mão na Massa — **34 – Aplicação de Backup/Restore no Firebird**
[*Daniel Sobrinho Laporte*]

Boas Práticas

Minicurso — **42 – FastReport: Elaboração de Relatórios Passo a Passo – Parte Final**
[*Giuliano Scombatti Pinto*]



Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Clube Delphi

Ano 13 - 146ª Edição 2012 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral

Paulo Quicoli
pauloquicoli@gmail.com

Editor Técnico

Daniel Sobrinho Laporte
daniel.laporte@gmail.com

Equipe Editorial

Guinther Pauli
guintherpauli@gmail.com

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br

Giuliano Scombatti Pinto
giuliano@sygnux.com.br

Jornalista Responsável

Kaline Dolabella - JP24185

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/central
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

EDITORIAL

É mais do que evidente que vivemos em uma geração onde nunca houve um acúmulo de informação como o existente. A quantidade é tão grande que novos modelos de armazenamento de dados precisaram ser criados, por exemplo, o NoSQL. Mas toda essa informação não vale nada se nada pode ser extraído dali. Com isso em mente, o assunto BI, mineração de dados, cubos de decisão está no topo da estratégia para empresas como Microsoft e Google. A Microsoft, por exemplo, aperfeiçoou para a próxima versão do SQL Server as ferramentas de BI, deixando-as mais acessíveis aos leigos. Contudo, pensando um pouco menor, você desenvolvedor que possui um sistema em algum cliente, com certeza possui um acúmulo de informações considerável. O que fazer com isso? Você vai deixar esses dados apenas ocupando espaço em disco? Nesta edição o Giuliano apresenta uma forma de se aplicar os conceitos de inteligência em seus aplicativos, partindo da mineração ao cubo OLAP. Ele também continua com a parte final do mini curso sobre FastReport, agora mostrando recursos mais avançados como criação de variáveis, exportação de relatórios, criação dinâmica e relatórios CrossTable. Neste último recurso vocês verão que há um relacionamento com o primeiro artigo.

O Daniel apresenta uma ótima ideia: a automatização do backup e restore de bancos Firebird. Você vão me dizer: "isso não é novidade", contudo, a atenção deve estar voltada para a ferramenta desenvolvida. Você verá como criar sua própria ferramenta de backup, o que pode tornar a vida de seu cliente muito fácil.

Já o Fabrício apresenta um material sobre LiveBindings e suas expressões. O programador Delphi está tão acostumado a utilizar controles data-ware que muitas vezes não vê as grandes possibilidades que esse mecanismo novo para ligação de dados trouxe. Espero que esse artigo chame a atenção para suas funcionalidades e desperte o interesse no recurso.

Para quem está começando, ou ainda não viu nada sobre o assunto, o Giuliano apresenta de forma didática o uso de Generics e Hash Tables. Boa leitura.

Abraço a todos.

Paulo Quicoli
pauloquicoli@gmail.com
Editor Geral
quicoli.wordpress.com
twitter.com/pauloquicoli

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Paulo Quicoli - Editor da Revista
pauloquicoli@gmail.com

MVP

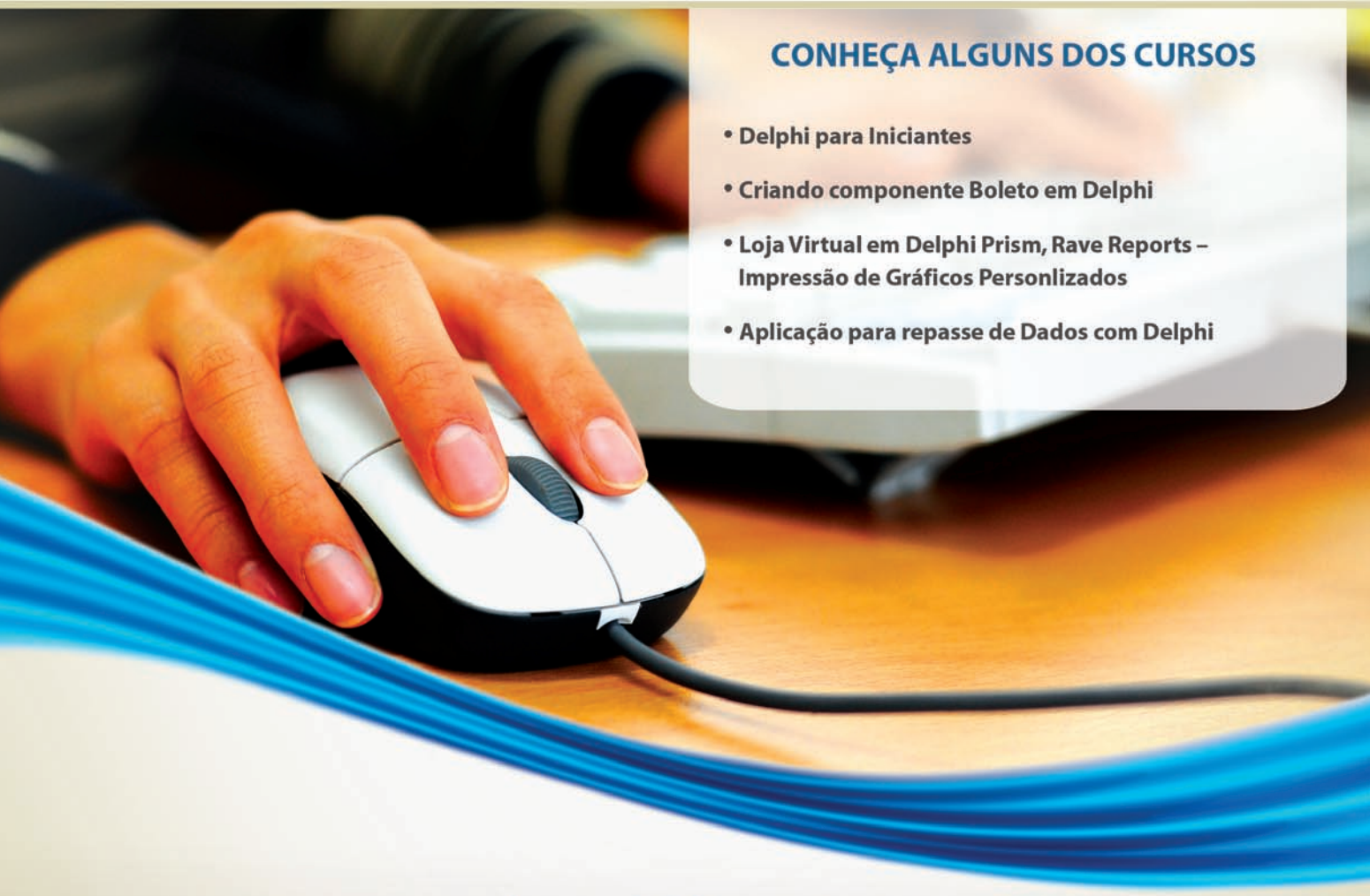
Assine agora e tenha acesso a todo o conteúdo da DevMedia: **www.devmedia.com.br/mvp**

DevMedia

CURSOS ONLINE

Clube Delphi

A Revista Clube Delphi oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.



CONHEÇA ALGUNS DOS CURSOS

- Delphi para Iniciantes
- Criando componente Boleto em Delphi
- Loja Virtual em Delphi Prism, Rave Reports – Impressão de Gráficos Personlizados
- Aplicação para repasse de Dados com Delphi

Para mais informações :

www.devmedia.com.br/cursos/delphi

(21) 3382-5038

http://www.devmedia.com.br

DevMedia

Mineração de Dados

Data Warehouse, Data Mining, BI e OLAP através do FastCube

Resumo DevMan

De que se trata o artigo:

Este artigo trata de apresentar uma introdução ao recurso de Data Mining, que permite a exploração de grandes volumes de dados para a obtenção de informações importantes previamente desconhecidas, abordando posteriormente os conceitos básicos de BI (Business Intelligence) e OLAP (On-Line Analytical Processing), enfatizando para isso a ferramenta FastCube, composta por diversos componentes e que traz uma metodologia para o emprego de OLAP e BI nas aplicações desenvolvidas.

Em que situação o tema é útil:

Este tema é útil em situações que se deseja empregar recursos importantes da atualidade para obter informações mais detalhadas através da utilização de recursos como Data Mining, BI e OLAP, e assim oferecer uma visão diferenciada do negócio para os gestores, para aplicações de grande e pequeno porte, além de enriquecer o software com funcionalidades interessantes, como é o caso do emprego da ferramenta FastCube.

Mineração de Dados :

Dentro do ambiente das corporações, a informação e o conhecimento são componentes chave e extremamente valiosos, afinal, muitas vezes o futuro e a saúde da organização dependem de decisões rápidas e coerentes, que por sua vez necessitam de uma base sólida. Ter à disposição ferramentas e estratégias que auxiliam a tomada de decisões na organização com base na informação armazenada é um fator extremamente importante e muitas vezes indispensável, onde tendências e probabilidades são apresentadas e podem ser fatores determinantes para a empresa em questão. Os modelos convencionais de consultas, relatórios e análise de dados podem não propiciar uma visão mais abrangente e qualificada da organização como um todo. Dentro deste cenário, existem diversas tecnologias, como é o caso do Data Mining, do BI (Business Intelligence) e do OLAP. O Data Mining, por sua vez, utiliza recursos para a busca de informações relevantes que estão presentes em meio a grande quantidade de dados através de diversas técnicas, informações estas que muitas vezes ficam despercebidas. O BI, inteligência nos negócios, com base nos dados, ajuda na tomada de decisões pelos gestores. Já o OLAP oferece uma diversidade de recursos para que a informação possa ser analisada de diversos ângulos, oferecendo funcionalidades como consultas e relatórios dinâmicos que podem ser elaborados pelos próprios usuários. Dentro deste contexto, o FastCube, desenvolvido pela FastReport, é uma ferramenta interessante e importante que auxilia o desenvolvedor que deseja empregar tais recursos em sua aplicação de uma forma rápida, agregando maior qualidade e funcionalidade a mesma. Este artigo tem por objetivo efetuar uma introdução aos recursos mencionados, apresentando assim a aplicação da ferramenta FastCube nos projetos, demonstrando um simples exemplo prático de como utilizar a mesma.

Notavelmente, dentro do contexto de aplicações direcionadas ao ramo comercial ou industrial, o banco de dados é o coração do sistema, armazenando toda ou a maior parte dos dados das instituições, bem como provendo todos os recursos básicos necessários para que os mesmos possam ser devidamente resgatados. Com o passar do tempo, em meio a tantos avanços tecnológicos acessíveis, enormes volumes de dados são acrescidos às bases, que muitas vezes dobram de tamanho em poucos anos ou em curtos períodos de tempo, onde é comum que antigas informações, contas e movimentos registrados acabem caindo "no esquecimento" e sendo pouco utilizados para uma análise mais profunda sobre a vida da organização, sem contar que as pesquisas e relatórios tradicionais podem não proporcionar a exibição e interpretação das informações com uma visão amplamente diferenciada, o que implica em muitos os casos no mau aproveitamento do conhecimento e entendimento para a alta administração. De fato, podem existir tendências e padrões implícitos nas informações armazenadas que poderiam propiciar maiores níveis de detalhamento da situação da organização aos gestores, além de apresentar constatações interessantes que

podem ser úteis em campanhas de marketing, por exemplo, como as probabilidades e características dos consumidores. Diante de fatores como estes, surge a necessidade de ferramentas e mecanismos que permitam que os dados possam ser analisados de uma forma otimizada em todos os sentidos, uma vez que os mesmos são de extrema importância, pois, armazenam toda a trajetória da instituição, auxiliando no processo de planejamento, análise de mercado, administração e apoio à decisão, onde tais informações estão escondidas ou não são percebidas em meio ao grande volume de dados. Um dos recursos básicos para o emprego de diversas tecnologias é o Data Warehouse (armazém de dados), que em síntese, é utilizado para armazenar conjuntos de dados organizados por assuntos, mantendo todo um histórico. Outro recurso da atualidade e um dos mais importantes quando o objetivo é a busca de conhecimento, é o Data Mining (Mineração de Dados). O Data Mining é um processo que consiste basicamente na identificação de informações relevantes que estão presentes em grandes bancos de dados, Data Warehouses ou repositórios, através de diversas técnicas, tais como associações, padrões consistentes, anomalias, estruturas e etc, sendo resultante da união de várias áreas, como é o caso da estatística e da inteligência artificial, e é uma ferramenta essencial para o processo de descoberta de conhecimento em base de dados, também denominado KDD (Knowledge Discovery in Databases). Quanto ao processo de Data Mining, o mesmo pode ser dividido basicamente em três etapas, sendo elas a exploração, a definição dos padrões e a validação dos dados.

Por permitir a extração de informações valiosas das bases de dados e apresentar uma nova visão de análise, possibilitando a definição de fatores estratégicos, a tecnologia Data Mining pode ser empregada para fornecer a base para diversos segmentos, como é o caso de Business Intelligence (BI). Em suma, o recurso de Business Intelligence ou Inteligência de Negócios pode ser definido como um conjunto de métodos e conceitos que podem ser implementados através de softwares com o intuito de utilizar os dados importantes da organização para auxiliar no processo de tomada de decisões, proporcionando melhorias para a alta administração e reunindo todas as informações preciosas em um único lugar. Sendo assim, esta ferramenta também tem o objetivo de transformar os dados em conhecimento, que por sua vez é utilizado para obter diversas vantagens em meio ao mercado altamente competitivo. Dentro de todo este contexto, outra tecnologia que existe para prover uma melhor e mais flexível análise das informações, é a OLAP (On-Line Analytical Processing ou Processamento Analítico On-Line). Ela permite uma visão conceitual de forma multidimensional das informações da organização, onde existem consultas que disponibilizam os dados relacionados a medidas, decompostas em diversas dimensões. Esta tecnologia permite que as informações sejam visualizadas e analisadas de diferentes perspectivas pelo usuário, mantendo toda a estrutura de dados adequadamente.

Dentro do contexto de visão multidimensional existem alguns termos que são empregados, como é o caso de Dimensão, Membro, Hierarquia, Medida e Cubo. O termo Dimensão faz referência a uma unidade de análise cujo objetivo é agrupar as informações de negócio que possuem relação, se tornando o cabeçalho de colunas e linhas. Já o termo Membro diz respeito a um subconjunto de determinada dimensão, onde uma dimensão pode ter vários membros. Em seguida, Hierarquia é a denominação adotada para definir todos os níveis que pertencem a uma dimensão, podendo a mesma ser ou não balanceada. Quando é balanceada, os subníveis são equivalentes e quando não, a equivalência hierárquica não ocorre. O termo Medida é adotado para definir uma dimensão que é especial e deve ser empregada na realização de comparações. A denominação Cubo é utilizada para identificar uma estrutura capaz de armazenar as informações de negócio de uma maneira multidimensional, possibilitando uma análise dos dados de uma forma facilitada e mais refinada.

Uma ferramenta interessante e completa para o emprego de OLAP nas aplicações desenvolvidas no Delphi é o FastCube, da FastReport. Esta ferramenta reúne uma diversidade de componentes que permitem ao usuário final, criar suas próprias consultas e relatórios utilizando os campos e valores que desejar de uma forma extremamente flexível e simples, podendo efetuar operações estatísticas como soma, média, mínimo, etc. O FastCube também pode ser utilizado em conjunto com as outras ferramentas desenvolvidas pela empresa, como é o caso do próprio FastReport, para impressão de relatórios e do FastScript, para a utilização de scripts para os mais variados objetivos. Sendo assim, o usuário consegue elaborar, editar, imprimir ou gravar suas consultas e relatórios facilmente. O FastCube também possui suporte a geração de diversos tipos de gráficos com base nas informações que estão sendo manipuladas, sendo outro ponto interessante desta ferramenta.

É possível notar que os recursos de Data Warehouse, Data Mining, BI e OLAP podem ser empregados em conjunto para fornecer, de maneira geral, uma melhor e mais adequada análise dos dados da organização, apresentando informações e novos pontos importantes para os gestores.

Data Warehouse

Em síntese, ele pode ser definido como um conjunto ou coleção de dados com base em assuntos, sendo devidamente integrado e de armazenamento permanente, utilizando variantes de tempo, oferecendo o suporte à tomada de decisões de uma maneira mais ágil. Normalmente, os Data Warehouses podem ser divididos nos mais diversos setores, tais como Marketing, para a avaliação de produtos e serviços dentro do contexto comercial, Financeiro, para o monitoramento das operações financeiras comerciais, Comportamental, onde os comportamentos e as informações dos clientes ficam armazenadas, e assim por diante.

Enquanto os bancos de dados tradicionais têm o objetivo principal de armazenar as informações das operações diárias da empresa, os Data Warehouses tendem a armazenar informações históricas guardadas por muito tempo. Vale à pena ressaltar que os Data Warehouses, através de ferramentas como ETL (ver **Nota do DevMan 1**), podem ser alimentados com os dados presentes nas bases de dados convencionais, ou até mesmo arquivos de texto, dentre outros.

Nota do DevMan 1

O processo de ETL (acrônimo de Extract, Transform and Load - Extração, Transformação e Carregamento) consiste basicamente na extração, conversão e carregamento dos dados de uma ou diversas fontes de origem, como bancos de dados, para determinada fonte de destino, que normalmente é um Data Warehouse. Os processos de extração e carregamento são necessários para o ETL, mas o processo de conversão ou limpeza das informações é um passo opcional. O ETL também é uma das fases mais críticas durante o processo de construção de um Data Warehouse, uma vez que pode demandar certo tempo e está alimentando o coração que será utilizado por outros recursos que necessitam de informações coerentes.

Para a criação de um projeto lógico de Data Warehouse, o processo de modelagem multidimensional é uma das técnicas empregadas. Nesta técnica, é elaborado o esquema estrela com base em um esquema conceitual criado durante a fase de análise. Dentro deste contexto, existem alguns conceitos básicos, tais como as tabelas de fatos (cubos de decisão), as tabelas de dimensões e as métricas. As tabelas de fatos são as principais estruturas que compõem o modelo dimensional e apresentam as transações de negócios, armazenando linhas (medições ou métricas) que estão relacionadas a determinado processo ou assunto. As tabelas de dimensões representam as várias visões que os usuários podem utilizar para efetuar uma análise das métricas, sendo formadas por diversos atributos que permitem a filtragem e agrupamento das informações. Já as métricas são atributos de um fato que representa o comportamento de determinado negócio para a dimensão. No esquema estrela, um dos mais comuns e utilizados, existe a tabela de fatos e uma tabela para representar cada uma das dimensões. Um exemplo prático deste esquema seria a existência de uma tabela de fatos denominada "FatosVenda", constituída pelos campos IDCliente, IDVendedor, Quantidade e Valor. Sendo assim, existiria uma tabela dimensão denominada "DimensaoCliente", contendo o IDCliente dentre outros campos como Nome, e outra "DimensaoVendedor", contendo o IDVendedor, etc, vinculadas através de tais campos.

Data Mining

Como mencionado anteriormente, Data Mining é um recurso empregado quando o assunto é obter informações válidas previamente desconhecidas de grandes volumes de dados para auxiliar na tomada de decisões nas organizações ou ainda, pesquisas da área científica, dentre outros objetivos. Este recurso, por sua vez, reúne a aplicação de diversas técnicas, muitas vezes complexas, para a exploração e identificação dos dados informados, podendo determinar relacionamentos entre variáveis que anteriormente não possuíam relação, utilizando metodologias inteligentes para o conhecimento e a definição dos padrões. Também automatiza o processo de captura e análise das informações, permitindo a determinação de um significado, demonstrando as peculiaridades do passado e as probabilidades constatadas para o futuro, possibilitando assim que os gestores das instituições possam prever tendências e atuarem de uma forma mais eficiente em meio aos desafios encontrados no cotidiano.

Dentro do processo de Data Mining, existem algumas metas que podem ser adotadas para a extração do conhecimento, tais como a Análise de Agrupamento, a Análise de Séries Temporais, a Classificação, a Sumarização, as Regras de Associação, o Modelo de Dependência, os Modelos de Relacionamento entre Variáveis, dentre outras. A **Tabela 1** apresenta uma definição resumida para tais metas que podem ser empregadas.

Meta	Descrição
Análise de Agrupamento	Efetua uma associação entre um item e uma ou mais classes, onde tais classes são definidas de acordo com os dados. Também é conhecida como Cluster.
Análise de Séries Temporais	Efetua uma determinação de características sequenciais, onde o intuito é definir um estado do processo obtido e registrar possíveis variações e tendências com relação ao tempo.
Classificação	Determina uma classificação ou associação entre um item e uma ou diversas classes pré-estabelecidas.
Sumarização	Efetua uma descrição reduzida para determinado subconjunto.
Regras de Associação	Efetua a definição de relacionamentos entre os campos que compõe a base de dados.
Modelo de Dependência	Determina uma descrição para as dependências relevantes das variáveis.
Modelo de Relacionamentos entre Variáveis	Efetua uma associação entre determinado item e uma ou mais variáveis.

Tabela 1. Algumas metas que podem ser empregadas em Data Mining

Com relação à utilização prática das ferramentas de Data Mining, existem diversos softwares elaborados para este objetivo, tais como o Clementine, o RapidMiner, o Weka, o Tanagra, dentre outros, gratuitos e pagos. Dentro do contexto do Delphi, também existem componentes que podem ser adicionados para prover tais recursos, como é o caso do conjunto de ferramentas do DewLabStudio, desenvolvido pela Dew Research (ver seção **Links**). Entretanto, o desenvolvedor pode ainda criar sua própria ferramenta de Data Mining se assim desejar, podendo adotar os algoritmos e técnicas conhecidas em seu projeto. Vale à pena evidenciar também que muitos SGBDs (Sistemas de Gerenciamento de Banco de Dados), tais como o Microsoft SQL Server, possuem ferramentas que oferecem diversos recursos para o emprego de Data Mining com base em bancos de dados relacionais, como é o caso da ferramenta SQL Server Analysis Services (SSAS).

Este recurso também pode ser empregado nos dados armazenados em Data Warehouses, para assim permitir a obtenção de informações mais refinadas.

BI (Busines Intelligence)

O termo BI é empregado para definir softwares, metodologias, tecnologias, dentre outros recursos, que se baseiam em dados e acontecimentos para apresentar informações estratégias que oferecem auxílio à tomada de decisão, possibilitando uma melhor escolha para o procedimento que deve ser realizado diante de determinado fator, sendo um substituto ao termo SAD (Sistemas de Apoio à Decisão). BI também pode ser definido como a habilidade que uma aplicação tem em buscar e identificar informações desenvolvendo entendimento, conhecimento, tendências, entre outros, para o devido apoio aos gestores, utilizando uma diversidade de ferramentas para efetuar a coleta, o processo de análise e a resposta, possibilitando assim vantagens significativas competitivas para as organizações. Assim como no processo de Data Mining, as ferramentas de BI também tendem a ser baseadas nos Data Warehouses, que por sua vez, oferecem a estrutura básica para as mesmas.

Outro ponto importante com relação ao BI, é que além de oferecer diversas vantagens e auxílio aos gestores e alta administração, também permite que outros setores baseados em informações consistentes possam obter ganho, como é o caso do setor de compras de uma organização que pode determinar qual fornecedor é o melhor para a aquisição de certo recurso, evidentemente, desde que tal funcionalidade esteja implementada.

Um software BI é baseado praticamente em quatro fatores, sendo eles os dados, as informações, o conhecimento e a decisão. Os dados, por sua vez, compreendem textos, imagens, sons, fatos e etc. As informações são os resultados obtidos através da organização e processamento dos dados, onde os mesmos se tornam significativos e ganham utilidade. O conhecimento reúne conceitos, instintos, regras e etc, que oferecem a base para as decisões. Já a decisão é a escolha de uma ou mais alternativas para determinada ação que deve ser realizada com base nas informações e no conhecimento.

Quanto à implementação de BI nas organizações de um modo abrangente, o primeiro passo é evidentemente, a análise e o planejamento. Posteriormente, um estudo detalhado sobre o negócio e a definição do Design para o repositório de dados. Outro fator extremamente importante e necessário é a existência da matéria-prima coerente para o projeto, que neste caso são os dados armazenados que são originados dos processos operacionais das organizações, que normalmente estão armazenados em bancos de dados convencionais. Em seguida, podem-se utilizar ferramentas de ETL, conforme mencionado anteriormente, para alimentar um Data Warehouse e assim obter informações interessantes. O processo de Data Mining também é um fator que pode ser empregado em BI. Por fim, a implantação e o acompanhamento.

Outro recurso que pode ser empregado quando o assunto é Business Intelligence é o OLAP, que por sua vez será mais bem abordado posteriormente.

OLAP (On-Line Analytical Processing)

O termo OLAP é empregado para definir ferramentas que permitem a obtenção de respostas ágeis e flexíveis em consultas gerenciais com base em uma diversidade de dados de uma forma dinâmica, possibilitando que relatórios sejam manipulados e formatados pelo próprio usuário final, por exemplo, sendo um

recurso muito empregado em Business Intelligence. O OLAP pode ser adotado para análises financeiras, de marketing, dentre uma diversidade de áreas como é o caso de CRM (Customer Relationship Management – Gestão de Relacionamento com o Cliente). Uma ferramenta OLAP é extremamente importante, pois, permite que as informações possam ser analisadas através de múltiplas perspectivas, mantendo também uma estrutura eficiente e íntegra dos dados. Indo além de um software, uma aplicação que emprega este recurso é a solução para um ambiente, para a integração e a modelagem dos dados. A visualização das informações é realizada por meio de dados agregados, uma vez que tem por objetivo auxiliar os gestores no processo de tomada de decisões, apresentando as informações através de diversas dimensões muitas vezes hierárquicas e termos de medida através das tradicionais tabelas, ilustrando ainda o cruzamento de informações e geração de gráficos. Também emprega os conceitos de Drill Down, que por sua vez é a apresentação dos resultados de determinadas informações de uma forma mais detalhada, e Drill Up, que é o processo de resumir o detalhamento de determinada informação, além de oferecer suporte a diversos filtros, ordenação, etc.

Apesar de possuírem semelhanças, os recursos de OLAP e Data Mining possuem diversas divergências. A diferença principal é com relação à forma como a exploração dos dados é efetuada. Em aplicações OLAP, a exploração é baseada em verificações onde hipóteses podem ser elaboradas pelo desenvolvedor para serem confirmadas ou não. Já em softwares Data Mining, conforme foi mencionado, as informações são "garimpadas" de certa forma automática, sendo estudados e detectados os padrões para a busca de conhecimento, onde não existe ou existe parcialmente uma questão prévia. Entretanto, ambas podem ser empregadas em conjunto para aumentar os recursos da alta administração, onde o processo de Data Mining pode ser realizado primeiramente para a obtenção das informações sendo posteriormente base para o recurso de OLAP ser utilizado para uma análise mais refinada.

Os Data Warehouses também podem ser empregados como base para os softwares que utilizam este recurso, assim como nos processos de Data Mining e BI.

FastCube

Desenvolvido pela FastReports, oferece suporte à manipulação de grandes volumes de dados. Ele reúne uma diversidade de componentes para o Delphi XE 2 e versões anteriores, bem como outros ambientes de desenvolvimento de software, como é o caso do C++Builder e do Lazarus. Através dele é possível efetuar uma análise dos dados de uma maneira mais eficiente, onde a elaboração de tabelas com as informações resumidas também é suportada, oferecendo recursos para a criação de gráficos e relatórios de uma forma simples e ágil, que pode ser efetuada diretamente pelo desenvolvedor ou o próprio usuário final, sem exigir que tal usuário disponha de altos conhecimentos de programação. Os componentes do FastCube podem ser facilmente adicionados nas aplicações e interligados com as bases de dados através de componentes do tipo TDataSet, suportando padrões de conexão como o BDE, DBExpress, ADO, dentre outros.

Com relação às vantagens do FastCube, existe uma diversidade de funcionalidades, tais como a utilização de filtros, corte de dados de forma instantânea, operações estatísticas (média, mínimo, máximo, soma e etc), funções especiais, índices ilimitados para uma tabela de resumo, controles de visualização, totais, gravação de modelos, exportação para diversos formatos (HTML, XLS e etc), geração de gráficos com base na biblioteca TeeChart (ver **Nota do DevMan 2**), impressão, dentre outros recursos.

Nota do DevMan 2

O TeeChart é um componente desenvolvido por terceiros que acompanha o Delphi e provê aos desenvolvedores, a possibilidade de elaborar vários tipos de gráficos de uma maneira simples e rápida, utilizando um editor, onde os dados podem ser originados das mais diversas fontes, como é o caso de Arrays, DataSets e etc,

Existem alguns pontos importantes que também devem ser observados ao empregar o FastCube. Após sua integração com um TDataSet, por exemplo, todos os campos de tipos padrões para dados são suportados (Date, Time, String, Número, Boolean), porém outros tipos como BLOB, Memo, Gráficos, dentre outros, não são suportados e são automaticamente ignorados durante processo de inicialização da conversão dos dados.

Com relação às ferramentas que compõem o FastCube, existem diversos componentes para empregar as funcionalidades, como mencionado anteriormente. A **Tabela 2** apresenta os principais componentes do FastCube.

Outro ponto com relação ao FastCube é que o mesmo pode ser empregado em conjunto com o FastReport para impressão dos relatórios.

Componente/Classe	Descrição
TfcCube	Componente não visual principal responsável por armazenar os dados, sendo a base para a utilização da ferramenta.
TfcSlice	Componente não visual utilizado para armazenar o Layout dos dados e preparar as informações de acordo com a estrutura.
TfcGrid	Componente visual empregado para a exibição de uma Grid em conformidade com uma estrutura, permitindo a manipulação das informações pelos usuários.
TfcToolBar	Componente visual que disponibiliza vários botões para que o usuário possa efetuar diversas atividades com os componentes do cubo.
TfcChart	Utilizado para exibir os dados através de gráficos, sendo baseado na biblioteca TeeChart.
TfcChartToolBar	Componente visual composto por diversos botões que oferecem ao usuário funcionalidades para a realização de operações com o gráfico.
TfcGridReport	Componente que permite a impressão de um Grid, podendo exibir uma janela de pré-visualização.
TfrcCoponnents	Permite o registro do componente FastCube para o FastReport Designer.
TfrcCube	Componente utilizado no FastReport Designer com o objetivo de armazenar os dados para a elaboração de relatórios.
TfrcCrossView	Componente utilizado no FastReport Designer para a definição de uma estrutura e impressão de uma Cross Grid.
TfrcChartView	Componente utilizado no FastReport Designer para possibilitar a configuração das propriedades de um gráfico e sua impressão.

Tabela 2. Principais componentes do FastCube

O FastCube também oferece suporte ao emprego de funcionalidades do FastScript, outra ferramenta desenvolvida pela FastReports que, como seu próprio nome sugere, é empregada na elaboração de Scripts com suporte à diversas linguagens, como é o caso do Pascal Script.

Tutorial

FastCube na prática

Antes de iniciar um exemplo prático do FastCube, o mesmo deve ser obtido diretamente do site do FastReport (ver seção **Links**) e instalado. Após a conclusão da instalação, a aba FastCube 1.0 já pode ser acessada através do Delphi, entretanto, normalmente não estão presentes todos os componentes que serão necessários para este exemplo por padrão. Sendo assim, o FastCube necessita ser recompilado. Neste momento é importante salientar que o FastScript também deve ser obtido e instalado para que todas as funcionalidades que serão apresentadas neste exemplo possam ser executadas satisfatoriamente antes deste processo, bem como o FastReport, que por sua vez já acompanha a versão XE 2 do Delphi. Contudo, se não desejar recompilar, este exemplo ainda pode ser realizado parcialmente sem o emprego dos gráficos e dos Scripts. Com o FastScript e o FastReport instalados, para dar início ao processo de recompilação do FastCube basta acessar o diretório onde o mesmo se encontra ("C:\Program Files (x86)\FastReports\FastCube") e executar o programa denominado "recompile.exe". Posteriormente, ele deve ser configurado para que o processo de recompilação possa ser efetuado, lembrando que o Delphi deve estar fechado. A **Figura 1** apresenta a janela de configuração do recompilador do FastCube que foi definida para este exemplo.

É possível observar na **Figura 1** que foram definidos os suportes a biblioteca TeeChart, para a geração de gráficos, ao FastReport 4, a ferramenta responsável pela geração de relatórios, e ao Fast

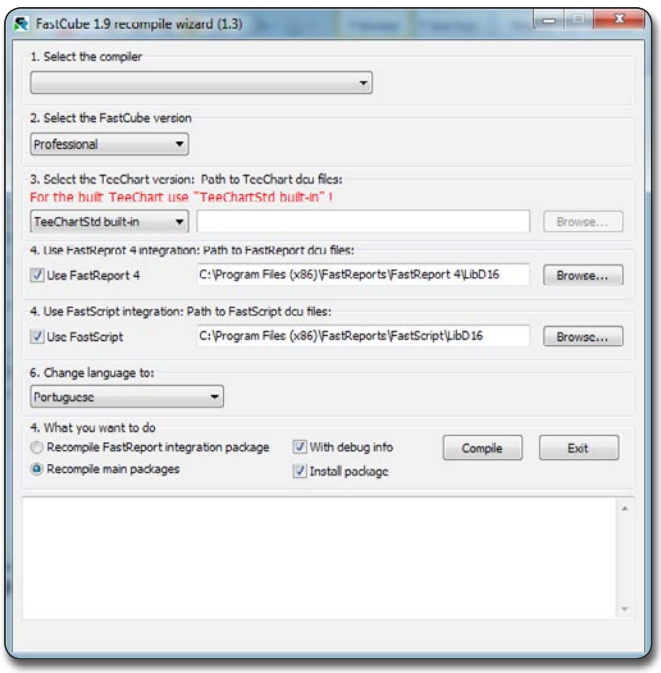


Figura 1. Janela de configuração do recompilador do FastCube

Script, que por sua vez oferece suporte a criação dos Scripts. Vale à pena evidenciar que a biblioteca TeeChart já acompanha o Delphi, então basta definir o valor padrão. Como pode ser observada também, a linguagem padrão foi alterada para Português, o que torna a ferramenta mais fácil de ser utilizada pelo usuário final na elaboração de suas consultas e relatórios.

Terminada a recompilação com o assistente, é criada uma nova pasta no diretório do FastCube contendo os novos pacotes. Neste momento, os novos pacotes precisão ser compilados através do Delphi, lembrando que os diretórios onde estão os recursos do FastCube e a nova pasta criada devem estar presentes no Library Path (Tools>Options>Library>Library path). A **Figura 2** apresenta os antigos e os novos componentes que foram adicionados após todo o processo de recompilação.

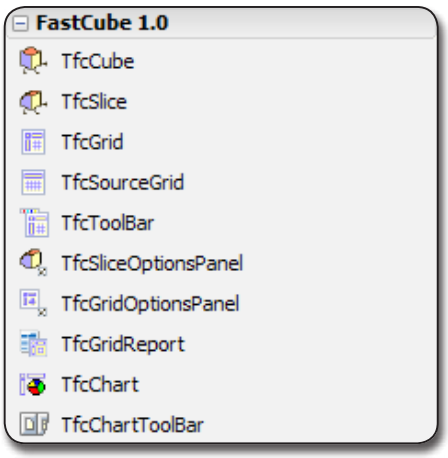


Figura 2. Aba de componentes do FastCube após a recompilação

Para dar início a um exemplo prático da utilização do conjunto de componentes do FastCube para a análise de informações utilizando OLAP, um novo projeto VCL deve ser iniciado (File>New>VCL Forms Application). Neste exemplo o formulário principal recebeu a denominação de FrPrincipal. O banco de dados utilizado para demonstrar os recursos foi o "dbdemos.mdb", presente no diretório "C:\Users\Public\Documents\RAD Studio\9.0\Samples\Data\", que por sua vez é um banco de dados de demonstração baseado no Microsoft Access que acompanha o Delphi e já contém diversas informações que podem ser utilizadas para exemplos e testes, sendo ele empregado neste projeto apenas para dar uma ideia de como utilizar a ferramenta FastCube. Para prover a conexão com a base será adotado o Framework ADO (ActiveX Data Objects), entretanto vale à pena lembrar que o desenvolvedor também pode utilizar ODBC (Open DataBase Connectivity), através da ferramenta de administração presente no Painel de Controle (Fonte de Dados ODBC), podendo ainda empregar o BDE (Borland Database Engine) ou uma solução de terceiros. Sendo assim, dois componentes da aba dbGo devem ser inseridos, um TADO Connection e um TADOTable. O TADOConnection é responsável por prover a conexão com o banco de dados baseado em Access, dentre outros, e o TADOTable, utilizado para acessar determinada tabela presente na base. Para dar continuidade, deve-se efetuar a devida configuração da propriedade ConnectionString do objeto ADOConnection1, lembrando que tal procedimento pode ser realizado através de um clique duplo sobre o mesmo. Com a janela de definição da ConnectionString aberta, deve-se clicar no botão Build. Na primeira aba (Provedor), basta selecionar "Microsoft Access 12.0 Database Engine OLE DB Provider", lembrando que tal provedor pode ser diferente dependendo da versão instalada do Microsoft Access ou do Windows. Em seguida, na aba denominada "Conexão", basta digitar o endereço completo da base de dados que será empregada, dbdemos.mdb, conforme mencionado anteriormente, e clicar no botão "Testar Conexão" para verificar se a comunicação é realizada. A **Figura 3** apresenta a janela de configuração da ConnectionString do objeto ADOConnection1 devidamente definida para este projeto.

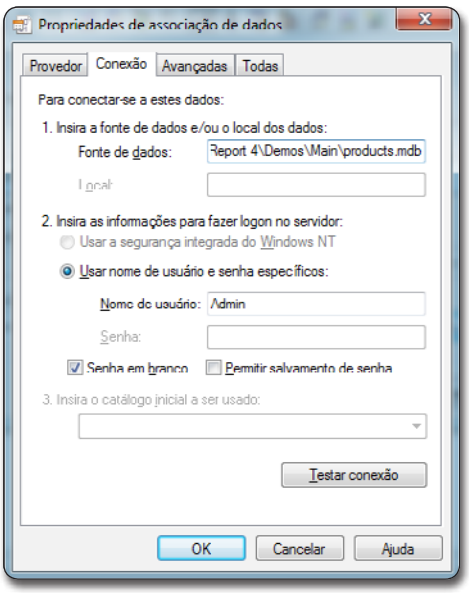


Figura 3. Janela de configuração da ConnectionString do ADOConnection1

Posteriormente, basta clicar no botão OK para finalizar a configuração da conexão. A propriedade Connected do ADOConnection1 foi definida como True, fazendo com que a conexão seja iniciada no momento da execução da aplicação. Também foi definida a propriedade LoginPrompt com o valor False para que a janela de autenticação de usuário e senha do banco de dados não seja exibida, uma vez que tais informações já foram definidas.

O próximo passo é configurar o objeto ADOTable1 adicionado anteriormente. A propriedade Connection do mesmo deve receber o valor de "ADOConnection1". Posteriormente, a propriedade Table deve ser definida com o valor de "orders", uma tabela que já possui informações de demonstração sobre vendas, e para finalizar, Active deve ser configurada como True, fazendo com que os registros da tabela sejam carregados.

Concluídas as definições dos objetos para a conexão com o banco de dados e a tabela, deve-se inserir no formulário principal um objeto do tipo TPageControl. Sua propriedade Align foi definida com o valor de alClient para que o mesmo ocupe toda a área visível do formulário, onde posteriormente foi adicionada mais uma página (clicando de direita e selecionando New Page) através do menu de contexto do mesmo, com o intuito de separar a futura apresentação do Grid e de um gráfico.

A próxima etapa para este exemplo é inserir e configurar os componentes do FastCube. Para tanto, um objeto do tipo TfcCube deve ser adicionado ao formulário, lembrando que ele é o componente base para a utilização dos recursos da ferramenta, sendo responsável pelo processamento Front-End e armazenamento dos dados. Sua propriedade DataSet deve ser definida com o valor de ADOTable1, interligando o mesmo a tabela do banco de dados. Vale à pena ressaltar também que ele pode ser utilizado diretamente com um arquivo especial de armazenamento de informações do tipo Cubo, cuja extensão é .mdc. Este componente também possui outras propriedades e subpropriedades interessantes que podem ser alteradas, como é o caso de DefaultFormat, onde o desenvolvedor pode definir os parâmetros para campos do tipo Moeda, Data, dentre outros. Em seguida, insira no formulário um TfcSlice. Este é responsável por conter o layout dos dados e suas relações com a estrutura (esquema), bem como indicadores de desempenho e diversos filtros, sendo um intermediário entre o componente TfcCube e os demais recursos da ferramenta. Após inserido, sua propriedade Cube deve ser definida com o valor de um objeto derivado de TfcCube, que neste caso é o fcCube1. Os objetos do tipo TfcSlice possuem outras diversas propriedades interessantes que convém defini-las, tais como DefaultColWidth e DefaultRolHeight, por exemplo, que permitem estabelecer a largura e a altura para as colunas e linhas, dentre outras.

Terminada a configuração básica para o funcionamento do FastCube, é o momento de adicionar os componentes visuais ao formulário para permitir aos usuários, além da visualização, a possibilidade de manipular as informações. O próximo componente inserido no formulário foi o TfcGrid, que por sua vez é um recurso interativo utilizado para representar os dados multidimensionais através de colunas e linhas em um plano 2D, apresentando os cabeçalhos de uma forma hierárquica. Através dele, os usuários podem adicionar ou remover campos, filtrar as informações, dentre outras diversas funcionalidades. Para seu devido funcionamento, o mesmo deve ser interligado a um objeto do tipo TfcSlice através de sua propriedade Slice, onde neste exemplo o valor é equivalente a Slice1. Em seguida, insira no formulário um componente do tipo TfcToolBar. O intuito deste componente é permitir aos usuários, mais possibilidades para manipular as informações, tais como adicionar funcionalidades estatísticas como

soma e média, ajustar a ordenação dos dados, salvar o esquema, carregar um esquema previamente gravado, exportar para outros formatos de arquivo (HTML/XLS/DOC/XML), salvar o cubo em disco rígido, dentre outras funcionalidades essenciais. Para que o mesmo possa ser devidamente utilizado, sua propriedade Grid deve ser definida com o valor de um objeto do tipo TfcGrid, tal como o fcGrid1 adicionado anteriormente. Este componente também possui uma diversidade de propriedades e subpropriedades que permitem a definição de sua aparência, bem como a atribuição das características visuais dos botões, dentre outras funcionalidades e possibilidades de personalização.

Neste momento, se o desenvolvedor definir a propriedade Active como True para o objeto fcCube1, a aplicação já pode ser executada e os dados serão carregados durante a inicialização da mesma. É perceptível que, sem nenhum trecho de código, já é possível iniciar a aplicação OLAP, entretanto, como não foram definidos nenhum esquema, cabe ao usuário final adicionar cada campo e/ou função que desejar, para assim poder obter as informações almejadas. Diante deste fator, o desenvolvedor pode, através de código-fonte ou do SliceEditor, pré-definir colunas e valores para que alguma informação seja carregada e visualizada no momento da criação ou exibição do formulário, tornando a tarefa do usuário mais simples. Para atingir tal objetivo, basta utilizar os diversos métodos que fazem parte do componente TfcSlice, onde um que merece destaque é o AddFieldTo. Este método permite que seja adicionada uma dimensão, ou em outras palavras, um campo da tabela/fonte de dados, dentre outros parâmetros, para uma região como o eixo Y ou X. Sendo assim, a **Listagem 1** apresenta um trecho de código que foi definido no evento OnCreate do formulário principal para que no momento da criação do mesmo, algumas colunas e informações possam ser exibidas automaticamente. A Unit fcTypes deve ser adicionada na seção Uses do formulário para prover alguns recursos que estão sendo empregados.

Listagem 1. Evento OnCreate(do FrPrincipal para carregamento do esquema

```
01 procedure TFrPrincipal.FormCreate(Sender: TObject);
02 begin
03   fcCube1.Active := True;
04   fcSlice1.BeginUpdate;
05   fcSlice1.AddFieldTo('SaleDate', 'Data Venda', rf_CapYAx);
06   fcSlice1.AddFieldTo('CustNo', 'Cliente', rf_CapYAx);
07   fcSlice1.AddFieldTo('ItemsTotal', 'Total', rf_CapFacts, af_Sum);
08   fcSlice1.AddCalcFieldTo(
09     'begin '+
10     ' if Measures["Total"].CurrentValue <> 0 then '+
11     '   Result := Measures["Total"].CurrentValue * 0.75'+
12     ' else ' +
13     '   Result := 0; '+
14     'end.', 'Custo (75%)', 1, af_Formula, '');
15   fcSlice1.AddCalcFieldTo(
16     'begin '+
17     ' if Measures["Total"].CurrentValue <> 0 then '+
18     '   Result := Measures["Total"].CurrentValue * 0.25'+
19     ' else'+
20     '   Result := 0; '+
21     'end.', 'Lucro (25%)', 2, af_Formula, '');
22   fcSlice1.AddFieldTo(sMeasuresFieldName, '', rf_CapXAx);
23   fcSlice1.EndUpdate;
24 end;
```

Em um primeiro momento, na linha 3 é possível notar que o objeto fcCube1 é ativado, lembrando que os objetos ADOConnection1 e ADOTable1 também devem estar ativos (propriedade Active). Em seguida, o método BeginUpdate do fcSlice1 é chamado (linha 4). Este método é responsável por dar início ao processo de atualização da estrutura, não permitindo que outras operações em paralelo sejam efetuadas para a mesma. Posteriormente, na linha 5 o método AddFieldTo é empregado para adicionar um dos campos que será utilizado, "SaleDate", onde o primeiro parâmetro é o próprio nome do campo, o segundo, seu rótulo (Caption), e o terceiro, a região onde o mesmo será adicionado, onde neste caso é o eixo Y (rf_CapYAx), sendo o mesmo representado por uma coluna. Na linha 6, um processo semelhante é realizado, porém para adicionar o campo "CustNo", que contém o número do Cliente. Na linha 7, o método AddFieldTo é chamado novamente, porém agora ele é empregado para adicionar o campo "ItemsTotal", que armazena o valor total para as regiões (rf_CapFacts), onde são apresentados os resultados, utilizando também uma função agregada do tipo SUM (af_Sum). Para simular um cálculo manual de valor de Custo baseado em 75% e um Lucro baseado em 25% com relação ao total, é empregado um novo método, o AddCalcFieldTo. Este método permite a adição de uma nova região com base em um script e assim, possibilita a exibição das mais variadas informações, como é o caso de um cálculo mais específico, lembrando que o FastCube necessita do FastScript para a realização desta etapa, que por sua vez foi habilitado no momento da recompilação dos componentes. Neste pequeno trecho de script, após a inicialização (Begin) presente na linha 9, é efetuado um teste para verificar se a medida denominada "Total", adicionada anteriormente na linha 7, possui seu valor atual equivalente a zero (linha 10). Caso a sentença seja verdadeira, então o resultado para esta coluna será o valor presente

na medida Total multiplicada por 0.75, para a exibição de um valor de Custo de exemplo (linha 11), senão o resultado será zero (linha 13). Na linha 14, o script é finalizado (end.), sendo também atribuído um rótulo para a nova região "Custo (75%)", a posição da mesma ("1") em relação as colunas, o tipo de função (af_Formula), e um último parâmetro vazio cujo objetivo seria a inserção de um filtro. Em seguida, da linha 15 até a linha 21, o mesmo método (AddCalc FieldTo) é utilizado com um script de forma semelhante, porém agora para calcular o Lucro de 25% sobre o total e adicioná-lo na região 2. Na linha 22, o método AddFieldTo é utilizado novamente, porém agora para adicionar a medida (sMeasuresFieldName), que neste caso é o Total, para o eixo X, através do parâmetro rf_CapXAx. Para concluir, o método EndUpdate é chamado para finalizar a atualização da estrutura que foi iniciada com o método BeginUpdate.

Como foi observado, através de um simples trecho de código fonte, o desenvolvedor pode pré-definir os campos que deverão ser exibidos, bem como atribuir os eixos que tais campos pertencem, podendo ainda adicionar cálculos manuais através de Scripts, criando assim uma tabela resumida. Entretanto, se não quiser escrever o código fonte para definir os campos, o desenvolvedor ainda conta com o SliceEditor do componente TfcSlice, mencionado anteriormente, que pode ser facilmente aberto através de um clique duplo sobre o mesmo. Por meio dele, o desenvolvedor também pode pré-definir quais são os campos e informações que serão apresentadas no momento em que o objeto do tipo TfcSlice for ativado. Também é possível definir a propriedade StoreInDFM do objeto TfcSlice em questão com o valor de True, para que as informações sejam armazenadas no arquivo DFM do formulário (ver **Nota do DevMan 3**).

Uma funcionalidade importante que será adicionada neste momento ao projeto é permitir aos usuários a impressão das informações que foram manipuladas e definidas no fcGrid1. Para tanto, basta adicionar ao formulário, o componente TfcGrid Report. Ele é responsável por exportar a Grid para o FastReport e assim possibilitar a pré-visualização e impressão da mesma. Sua propriedade denominada Grid deve ser definida com o valor de fcGrid1, efetuando a ligação entre ambos. Para finalizar o processo de impressão, um TSpeedButton foi adicionado ao formulário, onde sua propriedade Caption recebeu o valor "Imprimir" e seu evento OnClick, o trecho de código apresentado a seguir:

```
fcGridReport1.PrintPreview;
```

Através deste método, a janela de pré-visualização e de impressão do FastReport é exibida contendo a Grid definida e a mesma pode ser facilmente impressa.

Para adicionar mais funcionalidades à aplicação, neste momento serão inseridos os recursos necessários para a elaboração de gráficos através do FastCube e assim apresentar as informações multidimensionais de um novo ângulo. Para adicionar tal funcionalidade, um componente do tipo TfcChart deve ser inserido no formulário, lembrando que o mesmo é baseado na biblioteca TeeChart. Neste exemplo, por questões de organização, ele foi adicionado na segunda página do objeto PageControl1 criado anteriormente. Para utilizá-lo, basta definir sua propriedade Slice com o valor de Slice1, efetuando a interligação com o mesmo. Para que o gráfico possa ser exibido, a propriedade Active do objeto fcChart1 deve ser definida como True. Contudo, para ter mais opções na exibição e manipulação dos gráficos através do FastCube, também foi adicionado um componente do tipo TfcChartToolBar. De forma semelhante ao TfcToolBar, o TfChartToolbar também contém uma suíte de botões e recursos, porém destinados ao gráfico, permitindo gravá-lo, copiá-lo, alterar seu estilo (barras, linhas, pontos, etc), dentre outras funcionalidades. Para interligá-lo ao devido gráfico basta definir sua propriedade Chart, onde neste exemplo o valor atribuído foi fcChart1.

Neste momento, este exemplo encontra-se concluído. A **Figura 4** apresenta o visual da Grid com base nas informações

Nota do DevMan 3

Arquivos do tipo DFM são utilizados para armazenar as informações relacionadas aos componentes presentes nos formulários em uma aplicação desenvolvida no Delphi, tais como as propriedades. Vale à pena ressaltar que o mesmo é automaticamente atualizado quando o desenvolvedor adiciona ou remove componentes, bem como altera propriedades. Por ter seu conteúdo baseado em texto, este arquivo também pode ser alterado diretamente, mas normalmente não é uma tarefa necessária.

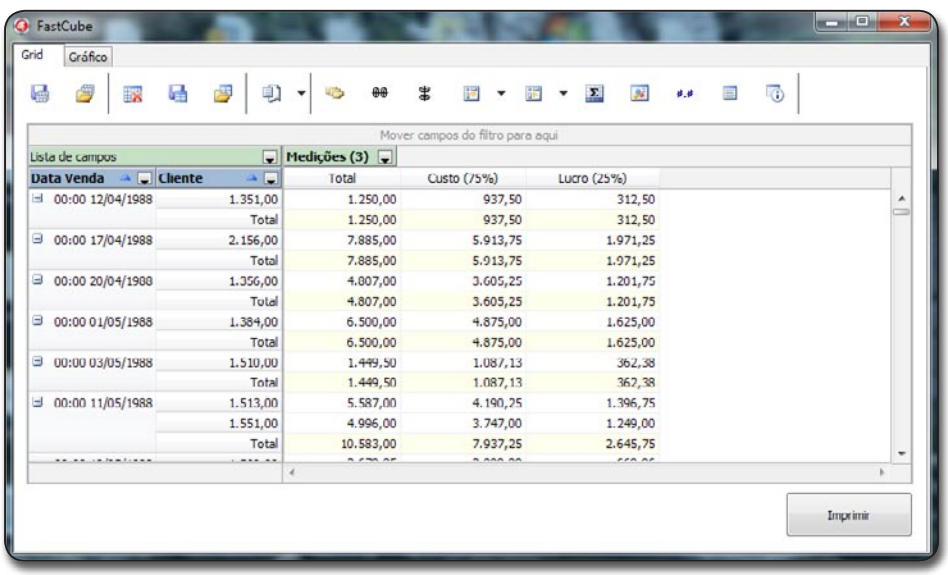


Figura 4. Grid com algumas informações da venda

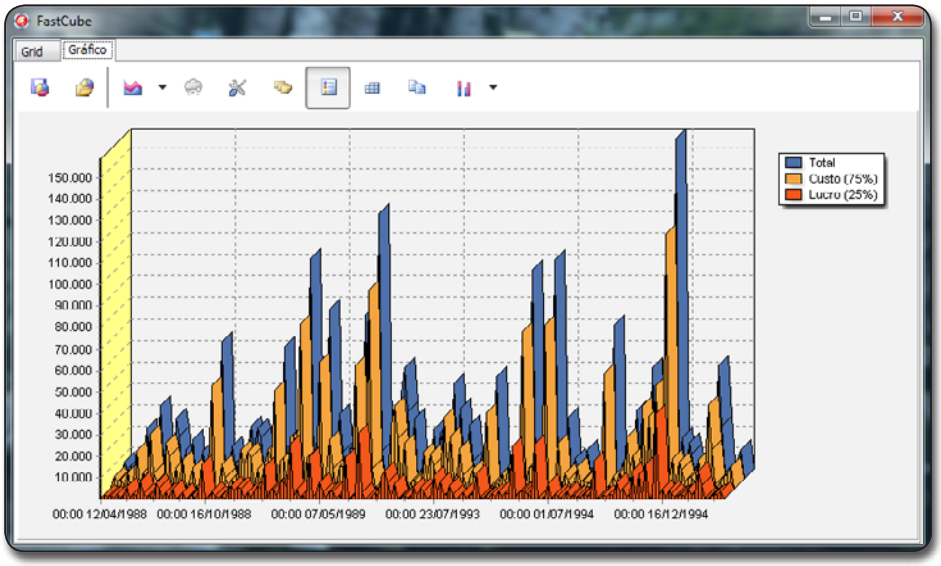


Figura 5. Gráfico gerado com base nas informações

de exemplo e a **Figura 5** apresenta a exibição de um gráfico gerado pelo FastCube com base nas informações.

Além dos componentes apresentados, o FastCube também conta com o TfcSliceOptionsPanel e TfcGridOptionsPanel, dentre outros, que são empregados para que o usuário final possa fazer ajustes em propriedades do TfcSlice e do TfcGrid, bem como a aplicar de filtros, onde basta interligar os mesmos aos seus devidos componentes.

Como pode ser observado, o FastCube tem diversos recursos para o emprego de OLAP que podem ser utilizados de uma forma extremamente simples, sem a obrigatoriedade de códigos pré-definidos, e bem flexível para os usuários finais, que por sua vez, podem criar e imprimir seus próprios relatórios.

Conclusão

Os dados de todos os processos de negócio são um dos maiores e mais importantes bens que as organizações detêm em suas mãos, guardando valiosas fontes de conhecimento e tendências. Entretanto, apenas possuir tais dados e efetuar consultas e relatórios tradicionais não garantem que as mesmas sejam utilizadas ao máximo para a apresentação de informações qualificadas e refinadas. Sendo assim, as empresas que pretendem dar um passo a frente, identificando o comportamento de seus clientes e prevendo futuras situações para que as melhores decisões sejam tomadas, necessitam de ferramentas que possibilitem uma melhor análise dos dados, descobrindo informações previamente não visíveis, e transformando tudo em conhecimento. Os recursos de Data Mining, Business Intelligence e OLAP apresentados neste artigo, são ideais para serem empregados dentro de cenários como este, oferecendo o devido apoio a alta administração e departamentos vitais para a saúde da empresa. Também é possível observar que tais recursos podem ser empregados em conjunto para um melhor resultado na busca do conhecimento.

Como apresentado neste artigo, uma ferramenta OLAP importante que pode e deve ser empregada não só em aplicações de grande porte, mas também em softwares de pequeno porte é o FastCube. Com seus diversos componentes e funcionalidades, esta ferramenta permite que os próprios usuários criem suas consultas, agrupamentos e relatórios rapidamente, adicionando funções estatísticas de uma maneira muito fácil, possibilitando ainda a impressão e gravação, sem contar que disponibiliza a elaboração e a exibição de gráficos, que por sua vez, tendem a dar uma visão mais interessante e destacada das informações. Sendo assim, o desenvolvedor conta com um importante conjunto de componentes para melhorar sua aplicação e adicionar diversas funcionalidades de uma forma rápida, sem a necessidade de escrever códigos e mais códigos para atingir seu objetivo.

Para concluir, atualmente existem várias ferramentas e recursos que auxiliam e muito as organizações nas tarefas futuras e do dia-a-dia além das apresentadas, e cabe as mesmas investir de acordo com suas possibilidades nos métodos mais adequados e indicados para que não sejam surpreendidas por contratempos em suas trajetórias, que por sua vez, poderiam ser evitados se as informações e probabilidades tivessem sido obtidas, analisadas e acompanhadas dos mais diversos pontos de vista.

Informações e Download do FastCube
http://www.fast-report.com/en/download/fast-cube-download.html

Informações e Download do FastScript
http://www.fast-report.com/en/download/fast-script-download.html

Site da Dew Research com componentes para Data Mining
http://www.dewresearch.com

Download do Enhanced Borland Decision Cube
http://sourceforge.net/projects/fxds/

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Seu **conhecimento vale muito** na DevMedia

Publique artigos e vídeos gratuitamente no site e participe do programa **PayContent.**

Sua participação pode ser premiada: quanto mais você postar, mais chances de ganhar.

Para saber mais acesse:
www.devmedia.com.br/paycontent

DevMedia

LiveBindings na prática

Entendendo expressões

O termo LiveBindings, se traduzido de forma livre para o português, tomará o significado de algo como "ligações ao vivo", o que no mundo do desenvolvimento não quer dizer muita coisa. O único indício que deixa transparecer é seu vínculo com algum recurso de Binding, neste caso Data Binding. De forma geral, o Data Binding consiste numa técnica que estabelece a ligação entre fontes de informações, a fim de se manter uma sincronia de dados entre as mesmas. A grosso modo, no Delphi este cenário é visto desde suas primeiras versões, principalmente em aplicações VCL Forms envolvendo banco de dados, refletido através da relação de uso entre DataSets e controles Data-aware (ver **Nota do DevMan 1**), os famosos componentes DB (TDBEdit, TDBText, TDBGrid, TDBNavigator, entre outros). De um modo prático, tomando como exemplo um TDBEdit, ao ligá-lo a um campo (TField) de um TClientDataSet por meio de um DataSource, automaticamente é estabelecida uma sincronia de dados entre eles. Dessa forma, os dados do campo são devidamente mostrados em tela, através do controle, bem como qualquer alteração sobre estes dados na UI é refletida no campo, caracterizando assim um Data Binding.

Nota do DevMan 1

No Delphi, o termo Data-Aware Components é designado para definir os componentes capazes de utilizar dados de outro componente, tal como um descendente de DataSet, por meio de um DataSource. Como exemplo, o controle TDBComboBox da VCL, nada mais é do que um controle combo Data-Aware, bem como um TDBCheckBox é considerado um controle Data-Aware de seleção de valores.

Com o lançamento do Delphi XE2, a Embarcadero introduziu então um novo conceito relacionado a ligações de dados, resultando em um novo recurso denominado LiveBindings. Em síntese, este mecanismo permite que qualquer propriedade de um objeto qualquer seja associada a uma expressão. Neste caso essa expressão pode ser, por exemplo, uma mera propriedade de outro objeto, outra propriedade do mesmo objeto, ou ainda uma expressão complexa que envolva desde operadores a métodos. Por esta razão o LiveBindings torna-se único, podendo ser aplicado nas mais variadas situações, numa amplitude maior de classes e propriedades compatíveis. Tomando como exemplo novamente a VCL, numa situação clássica do uso do LiveBindings, podemos ter então a associação entre dois controles, do tipo TProgressBar (ProgressBar1) e TSpinEdit (SpinEdit1). Sendo mais específico, suponhamos que haja a necessidade de se vincular a propriedade Position do primeiro com a propriedade Value do segundo, fazendo com que o incremento ou decremento de valores no SpinEdit seja refletido diretamente na posição da

Resumo DevMan

De que se trata o artigo:

O presente artigo trata de fundamentar o LiveBindings, uma das grandes novidades apresentadas pelo Delphi XE2 e que se caracteriza por ser um mecanismo voltado à criação de associações entre objetos. Por se tratar de um tema abrangente, o foco deste trabalho será sobre seus principais conceitos, expostos então da teoria à prática. Diz-se abrangente pelo fato do LiveBindings ser suportado por linguagens diferentes (Delphi e C++), bibliotecas de componentes diferentes (VCL e FireMonkey) e plataformas diferentes (Windows, MacOS e iOS), todos relacionados à família XE2 da Embarcadero.

Em que situação o tema é útil:

Para a criação de interfaces visuais que se conectam a dados, de forma mais dinâmica e responsiva, com o mínimo de esforço, em especial a objetos de negócio ou até mesmo criar vínculos entre controles visuais.

LiveBindings:

LiveBindings, FireMonkey e DataSnap Mobile Connectors são apenas algumas das várias novidades trazidas pela versão XE2 do Delphi. Sendo assim, um prévio estudo sobre cada uma delas se faz necessário para que o desenvolvedor se mantenha atualizado mediante ao que sua ferramenta de trabalho lhe provê em termos de recursos. Indo além, o aprofundamento em determinada novidade fica a cargo da necessidade e da disposição do profissional. Do ponto de vista do desenvolvimento de Software, a apresentação de uma nova tecnologia, produto ou recurso tem como início a exposição de seus conceitos envolvidos, sua pré-visualização de utilização e, finalmente sua utilização na prática. De forma a elucidar este cenário, o artigo se foca no LiveBindings, contextualizando-o de forma geral, da teoria à prática. Antecipadamente, podemos rapidamente defini-lo como sendo um novo recurso de Data-Binding (ligação de dados) suportado tanto pela tradicional VCL, como também pela nova plataforma FireMonkey.

barra de progresso. Anteriormente ao LiveBindings, o estabelecimento desta associação seria então refletida num processo todo manual, via código, tal como mostrado a seguir na **Listagem 1**.

Listagem 1. Mudança do ProgressBar através do evento OnChange

```
procedure TForm1.SpinEdit1Change(Sender: TObject);
begin
  ProgressBar1.Position := SpinEdit1.Value;
end;
```

Já a partir do Delphi XE2, com a utilização de seu novo mecanismo de Binding, podemos estabelecer essa mesma associação de uma forma mais direta, tirando proveito de um recurso apropriado para o processo, conforme será visto adiante. Indo além da VCL, talvez o maior beneficiário dos recursos providos pelo LiveBindings seja o FireMonkey. Isso pode ser justificado essencialmente pelo fato de que a biblioteca de componentes do FireMonkey não traz controles Data-Aware tal como a VCL. Dessa forma, utiliza-se o mecanismo de LiveBindings para estabelecer suas devidas associações de dados envolvendo seus componentes e possíveis fontes de informações, tal como um DataSet.

Conceitos

Fazendo uma prévia análise superficial, pelo fato do Live Bindings ter sido elaborado para atender tanto os cenários da VCL quanto o do FireMonkey, bem como disponibilizado tanto no Delphi quanto no C++ Builder, podemos interpretá-lo como sendo um mecanismo robusto. Dessa forma, é fácil presumir que sua parte conceitual como um todo é bastante extensa. Sendo assim, para compreendê-la de uma forma mais eficaz, nada melhor que se iniciar pelos seus conceitos mais básicos, elucidando situações mais triviais. Diante disto, podemos determinar que o uso mais natural do LiveBindings em aplicações, seja ela VCL ou FireMonkey, envolve basicamente três elementos: um objeto de origem, expressões e um objeto de destino.

Tomando como base uma simples aplicação Delphi envolvendo controles visuais (VCL Controls), o objeto de origem poderia ser traduzido como sendo o controle que irá prover informações ao objeto de destino. Este objeto de destino pode ser então outro controle, que irá receber as informações, mediante a expressão definida. Internamente no LiveBindings, estes conceitos são refletidos em propriedades, tais como ControlComponent, SourceComponent e Expressions (ControlExpression e SourceExpression).

Escopo

Quando começamos a estudar o tema LiveBindings no Delphi, é de praxe que o termo "escopo" esteja presente em grande maioria das literaturas a serem consultadas sobre o assunto. Dito isto, neste contexto podemos definir o termo escopo (scope) como sendo um conceito abstrato empregado para caracterizar tudo o que está ao alcance do mecanismo de expressão (expression engine) do LiveBindings. Entenda "alcance" como sendo aquilo que está visível e, portanto, disponível à Expression Engine. Sendo assim, durante o uso de LiveBindings, tipicamente há o envolvimento de dois escopos – escopo de entrada (Input Scope) e escopo de saída (Output Scope) – essencialmente refletidos nas propriedades ControlComponent e SourceComponent. Este cenário pode ser considerado o mais tradicional, como nas situações onde há o envolvimento de dois objetos, onde há a troca de informações de alguma de suas propriedades. Adicionalmente, o conceito de escopo ainda provê que a Expression Engine necessita de no mínimo um escopo para trabalhar, utilizando-o, nestes casos, tanto para Input quanto Output. Um caso típico é quando é realizada a vinculação de uma propriedade de um objeto à outra propriedade do mesmo objeto. Em situações como esta, partindo do princípio que o objeto em questão é um componente do Delphi, seja ele VCL ou FireMonkey, o mesmo representaria ao mesmo tempo tanto o ControlComponent quanto o SourceComponent no contexto.

Portanto, podemos perceber que a presença de um escopo torna-se fundamental ao conceito do LiveBindings. Em uma hipotética ausência de ao menos um escopo, seja ele de entrada ou de saída, a Expression Engine ficaria restrita a trabalhar com literais e operadores estáticos, perdendo-se toda a fundamentação do mecanismo.

Tutorial

Conceitos adicionais na prática

Para contribuir no entendimento do que foi falado até agora, vamos retomar ao exemplo mencionado anteriormente, onde se estabeleceu uma associação entre dois controles da VCL – Progress Bar e SpinEdit. Veremos agora como se daria essa relação através do uso de LiveBindings. Sendo assim, genericamente falando, o primeiro passo para a definição de um novo LiveBinding em uma aplicação VCL pode ser feita através da nova opção do menu de contexto disponível nos controles, conforme mostra a **Figura 1**. Elucidando a atual situação, o acionamento do menu poderia ser feito sobre o componente de destino, no caso o TProgressBar.

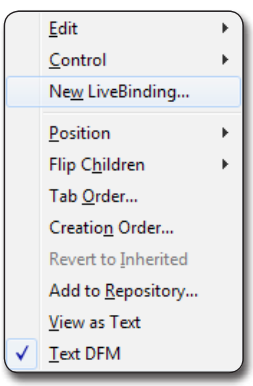


Figura 1. Opção de definição de LiveBindings no menu de contexto

Esta ação faz com que uma nova janela de opções seja exibida, conforme mostra a **Figura 2**. Neste momento deve-se selecionar então uma classe Binding, referente ao LiveBinding que se deseja criar. Estas classes são dividas em três grupos distintos, onde cada classe pode ser então interpretada como sendo um componente LiveBinding:

• **Binding Expressions**: envolve componentes que dependem da configuração de expressões para a efetiva determinação do Binding. Tais expressões são então avaliadas por um mecanismo de expressão interno, de forma a atribuir o valor resultante ao objeto alvo;

• **Links**: envolve quatro componentes que irão prover recursos de read/write de dados, ou seja, de leitura e gravação. Seu uso aliado com o componente TBindScopeDB provê a implementação de recursos Data-Aware a controles que não possuem esta característica, tal como os do FireMonkey ou mesmo um simples TEdit, da VCL;

• **Lists**: envolvem componentes considerados mais sofisticados, uma vez que são utilizados para se atribuir valores a expressões envolvendo Listas e Grids, tais como indicam suas nomenclaturas – TBindList e TBindGridList. Nestes tipos de Binding, especificamente, a ligação estabelecida é unidirecional, ou seja, o valor produzido só afetará a expressão de destino (ControlExpression), nunca a de origem (SourceExpression). Na prática, a utilização destes componentes, da mesma forma que os do grupo Links, envolve o uso de componentes do tipo BindScope (TBindScope e TBindScopeDB), cuja finalidade aqui é prover a ligação de dados a qualquer objeto que suporte estruturas de List ou GridList.

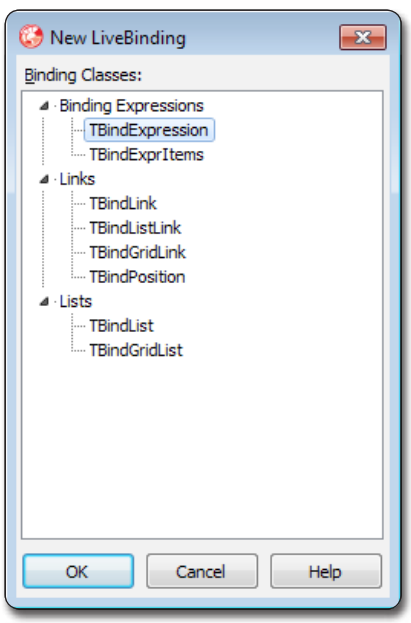


Figura 2. Opção de definição de LiveBindings no menu de contexto

Para o cenário do exemplo, o componente TBindExpression é o ideal, uma vez que o desejado é a atualização da expressão de destino a partir da expressão de origem. Uma vez que é feita a seleção da classe, um novo componente relacionado especificamente a LiveBindings é adicionado ao contexto do projeto. Trata-se do componente TBindingsLists, que será detalhado mais adiante no artigo. O importante neste momento do aprendizado é ter em mente que ele é peça fundamental para o processo, uma vez que ele funciona como uma espécie de gerenciador dos LiveBindings definidos. Na **Figura 3** é mostrado o Object Inspector, com as propriedades do objeto TBindExpression criado. Dentre elas, podemos ver as quatro propriedades citadas anteriormente. A seguir, é feita uma breve explanação sobre cada uma, tendo como base o contexto do exemplo proposto:

• **ControlComponent**: indica o objeto alvo que, para este caso, está representado pelo controle ProgressBar1;

• **ControlExpression**: expressão do componente alvo, ou seja, aquela que irá receber o valor da expressão do objeto de origem. Neste caso, representado pela propriedade Position do controle ProgressBar1;

• **SourceComponent**: indica o objeto de origem, que será o responsável por prover dados a ControlComponent por meio de sua expressão. Aqui é representado pelo controle SpinEdit1;

• **SourceExpression**: expressão do componente de origem que, para este caso, trata-se do valor da propriedade Value do controle SpinEdit1.

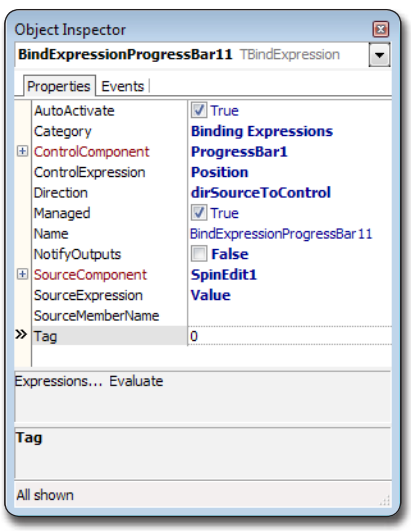


Figura 3. Propriedades relacionadas a LiveBindings

Conforme visto, ao se definir um Binding do tipo TBind Expression, é necessário que o mecanismo de expressão do LiveBindings faça a avaliação das expressões envolvidas. Para que isso ocorra, é necessário que se notifique o mecanismo sobre as alterações realizadas sobre a SourceExpression. Sendo assim, para este exemplo, isto poderia ser feito no evento OnChange do controle TSpinEdit, que é acionado toda vez que seu valor é alterado. Para isso, é feita uma chamada ao método Notifiy do componente TBindingsList, passando como parâmetros o próprio objeto tratado (Sender), e sua propriedade condizente à expressão (Value) (**Listagem 2**).

Listagem 2. Evento OnChange do SpinEdit com o notificador LiveBinding

```
procedure TForm1.SpinEdit1Change(Sender: TObject);
begin
  BindingsList1.Notify(Sender, 'Value');
end;
```

A **Figura 4** mostra a aplicação de exemplo em execução, onde a cada clique sobre as setas indicativas do TSpinEdit alterando o valor de sua propriedade Value, reflete no valor da propriedade Position do TProgressBar.

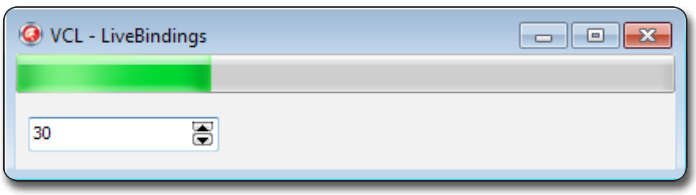


Figura 4. Aplicação de exemplo em execução

Um pouco mais sobre expressões

Conforme já foi dito e visto, a essência do LiveBindings envolve basicamente a criação de vínculos entre objetos e expressões. Partindo-se do princípio de uma situação trivial envolvendo dois objetos da VCL, uma vez associados, o que fundamentalmente irá determinar a transição de valores entre um e outro são as expressões. No contexto do mecanismo, essas expressões são então tratadas como Expressions. No popular, as Expressions nada mais são do que textos (Strings) que são avaliados por um motor que irá determinar os valores envolvidos na associação estabelecida. Com relação à String, esta pode ser constituída essencialmente de quatro elementos distintos – propriedades (de objetos), operadores, literais e métodos – conforme será visto adiante. Já com relação ao motor, este é então refletido num recurso interno denominado Expression Engine que avalia a expressão, produzindo uma saída com o resultado desta avaliação. Por conseguinte, o que o mecanismo do LiveBindings faz é atribuir este resultado a uma propriedade de determinado objeto presente na associação.

Dentre os possíveis elementos a serem utilizados em uma Expression, as propriedades (properties) de objetos podem ser consideradas as mais usuais, uma vez que a maioria das literaturas que apresentam o LiveBindings fazem uso deste elemento como exemplo. Citando novamente o exemplo apresentado anteriormente, temos a definição das expressões envolvidas (ControlExpression e SourceExpression) conforme mostra o trecho de código da **Listagem 3**.

Listagem 3. Definição das expressões envolvidas (arquivo .dfm)

```
object BindExpressionProgressBar11: TBindExpression
    Category = 'Binding Expressions'
    ControlComponent = ProgressBar1
    SourceComponent = SpinEdit1
    SourceExpression = 'Value'
    ControlExpression = 'Position'
    ...
end
```

Usando operadores (Operators)

Nas expressões do LiveBindings os operadores (ou Operators) são representados essencialmente por símbolos, que se tornam elementos-chave em situações onde há o envolvimento com literais. Sendo assim, sua utilização com literais Strings ou com literais numéricos podem ser designadas da seguinte forma:

```
//Uso com strings
"Largura do controle (Width): " + ToStr(Width);

//Uso com números
2 + 3 – (1 * 1) + (2/2)
```

No caso do uso como Strings, o operador "+" realiza a concatenação dos valores (texto) envolvidos, tal como já realizamos com Strings na programação Delphi. Da mesma forma, os operadores (+, -, * e /) utilizados com literais numéricos, tomam a forma de tradicionais operadores numéricos, causando o efeito de soma, subtração, multiplicação e divisão, respectivamente. Ainda neste contexto, outro detalhe a ser mencionado é que a Expression Engine permite que o desenvolvedor estabeleça priorizações no processo de avaliação de uma expressão complexa. Isto é conseguido pelo fato da Engine, durante a avaliação, considerar primeiro aquilo que está determinado entre parênteses. Sendo assim, a expressão 10 * (100 - 90) seria interpretada internamente como 10 * 10, tendo como resultado o valor 100. Por fim, o uso de operadores possibilita ainda a definição de expressões booleanas, através de operadores de comparação, tais como <, > e =. Num exemplo simplista, a expressão 10.1 > 10.0 resultaria num resultado booleano verdadeiro (True).

Usando propriedades (Properties)

A utilização de propriedades de objetos em expressões do LiveBindings é talvez a forma mais trivial de uso de Expressions. Isto porque, é de praxe numa abordagem inicial desejar demonstrar o vínculo direto estabelecido entre controles das bibliotecas de componentes do Delphi, tais como a VCL e o recente FireMonkey. Conforme já foi dito, por via de regra, as tratativas de Expressions no LiveBindings são providas basicamente por duas propriedades, denominadas SourceExpression e ControlExpression. Estas estão relacionadas diretamente a outras duas propriedades, SourceComponent e ControlComponent, que indicam o componente de origem e de destino, respectivamente. Dito isto, ao se utilizar propriedades em expressões do LiveBindings, é importante tomar como conceito o fato de que não é qualquer propriedade que poderá ser utilizada. Isto significa dizer que somente propriedades do objeto indicado em SourceComponent poderão ser utilizadas em SourceExpression. Da mesma forma, somente propriedades do objeto atribuído a ControlComponent poderão ser utilizadas em ControlExpression. Analisando mais cuidadosamente o conceito envolvido, podemos facilmente compreender este fundamento, uma vez que, caso não houvesse critério algum na determinação das propriedades utilizadas em SourceExpression e Control Expression, a Expression Engine do mecanismo de LiveBindings

poderia se perder. Em outras palavras, a partir do momento que se utiliza uma propriedade na expressão de ControlExpression, a Expression Engine já é direcionada a captar seu valor no objeto determinado em ControlComponent. Do contrário, numa exemplificação simples, uma vez que fosse informada em Control Expression uma propriedade comum a vários controles da VCL, tais como Height ou Width, a Engine não tomaria conhecimento de qual objeto (controle) exatamente esta se refere.

Usando métodos (Methods)

Da mesma forma que podemos afirmar que o uso de propriedades é a forma mais "introdutória" de se construir expressões no LiveBindings, podemos estabelecer que o uso de métodos talvez seja a forma mais flexível deste mesmo processo. Isto porque, neste âmbito, existem dois tipos de métodos que podem ser utilizados em expressões: os de objeto, tratados como Methods of Object e os personalizados, tratados como Custom Methods.

Os Custom Methods podem ser definidos como sendo métodos especiais e exclusivos para o uso em Expressions. Sendo assim, o próprio LiveBindings traz consigo uma série de métodos personalizados já definidos e prontos para serem usados pelo mecanismo. Adicionalmente, o desenvolvedor é capaz também de escrever seus próprios métodos utilitários para serem utilizados, contudo, na grande maioria dos casos, esta atividade é suprimida em virtude da vasta gama de opções de se construir expressões na tecnologia, tal como os próprios Custom Methods citados anteriormente. Na prática, o uso destes tipos de métodos numa expressão pode até causar certa confusão ou estranheza numa fase inicial, já que os escrevemos diretamente no valor de uma propriedade String. Todavia, uma vez que se adquire o pleno conhecimento de seu uso, o que vale ter em mente é que tais métodos serão devidamente avaliados pela Expression Engine do LiveBindings. Ao acionar a propriedade Methods do componente TBindingsList, podemos ver os métodos disponíveis para uso, conforme pode ser visto na **Figura 5**.

Methods of Objects, conforme sua nomeação indica, nada mais são do que os métodos de objetos, visíveis de alguma forma para o mecanismo de expressão (Expression Engine) do LiveBindings. Sendo assim, isso significa que tais métodos devem estar definidos com a visibilidade public ou published no escopo das classes dos objetos. Esses métodos assumem então a forma de procedimentos ou funções em que, internamente, o mecanismo de expressão espera que uma function retorne um tipo de dado condizente com a Expression, enquanto que uma procedure sempre retornará nil. Já os objetos podem ser os disponibilizados pelas classes da VCL ou FireMonkey, ou mesmo as próprias classes de negócio construídas pelo desenvolvedor. Para este último caso, a fim de chamar um método a partir de uma expressão, basta escrevê-lo nas classes relacionadas aos objetos definidos em Control Component ou SourceComponent do LiveBindings. Indo além, o uso de Methods of Objects permite ainda atingir certo grau de flexibilidade na hora de defini-los em uma expressão. Num exemplo clássico envolvendo o uso de controles da VCL, podemos definir métodos na própria classe do formulário onde esses controles estão inseridos e capturá-los nas expressões através do Owner do componente associado em ControlComponent ou Source Component. Neste caso, o proprietário (Owner) do componente será o próprio formulário.

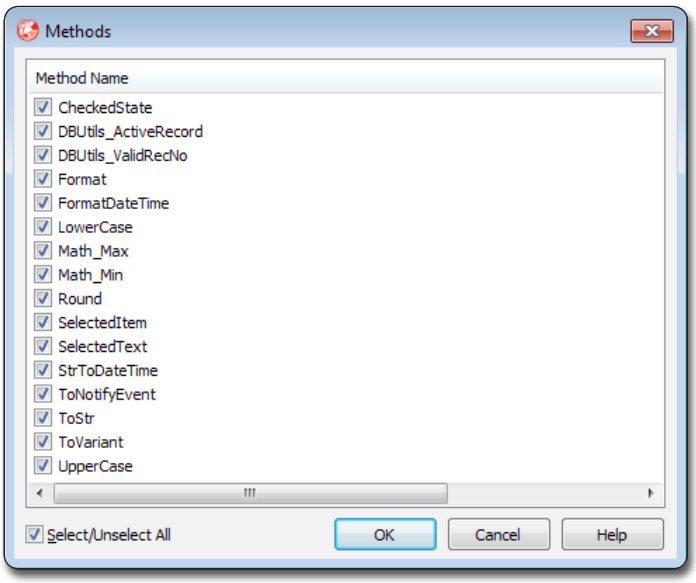


Figura 5. Métodos disponíveis

Usando literais (Literals)

O uso de literais nas expressões do LiveBindings significa basicamente o uso de Strings, de uma forma muito próxima da que é feita no código Delphi, com algumas diferenças. Essas dessemelhanças, no caso, estão relacionadas à forma com que a Expression Engine avalia tais literais Strings. Para facilitar este entendimento, podemos considerar que, tradicionalmente, o compilador Delphi reconhece um trecho de código como String, quando este está envolto por aspas simples (Single Quotes), tal como a seguir:

```
"Isto é uma string no Delphi"
```

Da mesma forma, durante a avaliação de uma expressão no LiveBindings, sua Expression Engine reconhece uma String da mesma maneira que o compilador Delphi, ou seja, tudo que estiver envolvido por aspas simples. O diferencial, neste caso, fica por conta do mecanismo também fazer este reconhecimento pelo uso de aspas (Double Quotes). Sendo assim, uma mistura de ambas as abordagens é válida, tal como o trecho a seguir:

```
'string um' + "string dois"
```

Ou ainda, a primeira expressão poderia ser reescrita a seguinte forma:

```
"Isto é uma string no Delphi"
```

Outro uso de aspas fica por conta de Strings que possuam apóstrofo. Para estes casos, o uso de aspas "duplas" faz com que se isole esteticamente o que é literal e o que é sinal de pontuação. Indo além, em meio a toda essa flexibilidade de uso de literais Strings em expressões do LiveBindings, uma ressalva importante deve ser feita: toda e qualquer String iniciada com aspas, deve ser terminada com aspas, o mesmo vale para o uso com aspas simples. Na prática, isso provê uma maior legibilidade do código:

```
'string um' + "string dois" + 'string um mais "string dois"'
```

Componentes da paleta LiveBindings

O LiveBindings, assim como a grande maioria dos recursos disponíveis no Delphi, é formado por uma agregado de classes que acabam por serem refletidas em componentes. Neste caso, a menção a "componentes" se refere tanto aos tradicionais componentes visuais, tais como os alocados na Tool Palette da IDE, quanto às "peças" reutilizáveis disponibilizadas pelo Framework. Dito isto, a partir do Delphi XE2, foi disponibilizado tanto para projetos VCL quanto FireMonkey uma nova gama de componentes localizados no grupo LiveBindings da paleta de ferramentas (**Figura 6**). Tais componentes podem então ser divididos em basicamente dois grupos: os que **suportam** LiveBindings e os que **implementam** LiveBindings. Mesmo que ambos estejam atrelados ao mesmo Framework, o primeiro grupo inclui componentes que não são considerados essencialmente LiveBindings, ao contrário do segundo, que engloba componentes considerados "nativos" e utilizados na criação efetiva de LiveBindings, tais como os vistos até aqui. Sendo assim, dando um enfoque agora nos componentes do primeiro grupo, temos os do tipo BindingsLists e BindScope (DB).

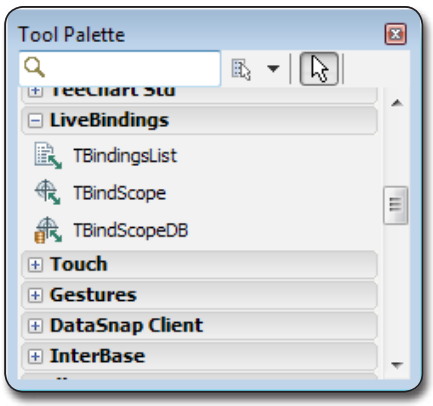


Figura 6. Grupo de componentes LiveBindings na Tool Palette

FÓRUM
DEVMEDIA
O lugar perfeito para você ficar por dentro de tudo o que acontece nas tecnologias do mercado atual

No fórum da DevMedia você irá encontrar uma equipe disponível e altamente qualificada com consultores e colaboradores prontos para te ajudar a qualquer hora e sobre qualquer assunto. Temos as salas de Java, .NET, Delphi, Banco de Dados, Engenharia de Software, PHP, Java Script, Web Design, Automação comercial, Ruby on Rails e muito mais!

ACESSE AGORA
www.devmedia.com.br/forum

DevMedia

BindingsList

O TBindingsList é o componente principal para se trabalhar com LiveBindings em tempo de Design. Isso se justifica pelo fato dele ser designado para funcionar como uma espécie de utilitário que irá gerenciar todos os eventuais Bindings criados em determinado formulário de uma aplicação. Sendo assim, através dele é possível ter acesso a cada LiveBinding estabelecido, bem como suas expressões associadas. A lista de todos esses Bindings e Expressions fica disponibilizada em uma estrutura tratada como sendo seu editor, conforme mostrado na **Figura 7**. Em meio ao contexto, sua utilidade é tão grande que muito provavelmente o desenvolvedor nunca irá precisar adicionar um TBindingsList manualmente em determinado ponto do projeto. Isto porque, na primeira vez em que se define um novo LiveBinding na aplicação, o próprio IDE se encarrega de adicionar automaticamente um TBindingsList ao cenário atual. Isto faz com que seu uso se torne natural, acentuando ainda mais sua importância.

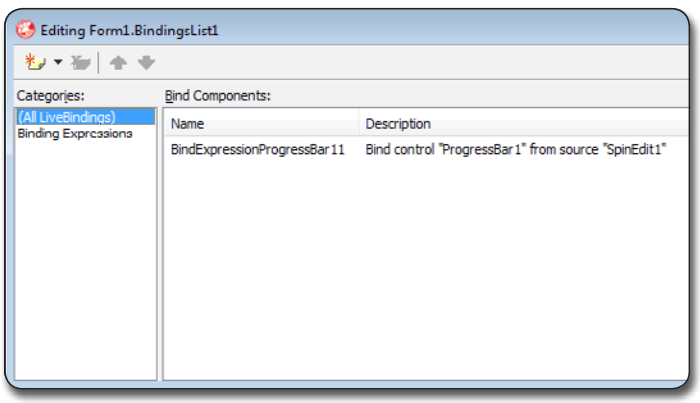


Figura 7. BindingsList exibindo o LiveBinding definido na aplicação de exemplo

BindScope

No contexto do LiveBindings, a adição de um escopo significa tornar possível a introdução de objetos, os quais poderão ter suas propriedades e métodos usados nas expressões eventualmente construídas. Neste sentido, são disponibilizados dois componentes específicos na Tool Palette para o provimento de escopo ao mecanismo de expressão do LiveBindings. O primeiro deles é o TBindScope que, em síntese, tem a função de fazer o Wrap de objetos em um único escopo mais geral. Sendo assim, para que uma Expression possa recuperar determinado membro do grupo de objetos envoltos pelo TBindScope, o componente suporta a interface IScopeEnumerator, onde cada membro é então um IEnumerable. Em seu uso prático, o TBindScope terá outros componentes LiveBinding apontados para si, estabelecendo uma espécie de link entre ambos. Isto resulta numa associação de comportamentos, principalmente no que diz respeito à ativação ou desativação de um Binding. Em outras palavras, a "ativação" do TBindScope faz com que todos os LiveBindings relacionados a ele também sejam ativados e, da mesma forma, quando da sua "desativação", todos os LiveBindings que o utilizam são desativados.

Todo este comportamento fica então condicionado à configuração de duas propriedades booleanas dos componentes LiveBindings: AutoActive e Active. A primeira, quando marcada como True, indica a possibilidade da ativação do componente ser feita pelo BindScope. Esta "ativação" nada mais é do que a atribuição do valor True à segunda propriedade citada, Active. Já o componente TBindScopeDB se liga a um DataSource tornando seus dados, provindos de uma fonte de dados tal como um DataSet, disponíveis a todos os objetos vinculados.

LiveBindings em aplicações FireMonkey

Apesar de já mencionado o suporte a LiveBindings no FireMonkey, até agora foram somente vistos cenários envolvendo aplicações VCL. Sendo assim, será visto agora como se dá o envolvimento deste novo mecanismo de Binding em aplicações FMX e o papel crucial exercido sobre elas, que não se limita a ligação de dados do banco a controles visuais. Conforme já visto anteriormente, uma das essências do LiveBindings é poder lidar com a interface de usuário das aplicações de uma maneira diferente do tradicional. Em outras palavras, através de sua utilização, o desenvolvedor dispõe de recursos para que a UI reaja aos estados dos controles em tela num processo mais "automático", sem a antiga necessidade de codificar manualmente processos deste tipo. Prova disso é que com o uso de LiveBindings, algo que antes dependia essencialmente de manipuladores de eventos agora pode ser feito através dos recursos de seu mecanismo. Em contrapartida, vale ressaltar que a intenção do LiveBindings não é o de substituir o uso de Event Handlers, mas sim o de aumentar a gama de recursos do Delphi para o desenvolvimento de projetos VCL e FMX.

Num exemplo trivial, poderíamos ter então dois componentes Label, que informam os valores dinâmicos de altura e largura do formulário ao qual estão inseridos. Dessa forma, o texto de cada Label seria atualizado conforme o Form é redimensionado pelo usuário. Na prática, define-se então dois objetos LiveBindings do tipo TBindExpression, tal como mostra o trecho de código da **Listagem 4**:

Listagem 4. BindExpressions dos componentes Label

```
object BindExpressionLabel11: TBindExpression
    Category = 'Binding Expressions'
    ControlComponent = Label1
    SourceComponent = Owner
    SourceExpression = #39'Largura do atual do Form: '#39' + ToStr(Width)'
    ControlExpression = 'Text'
    ...
end
object BindExpressionLabel21: TBindExpression
    Category = 'Binding Expressions'
    ControlComponent = Label2
    SourceComponent = Owner
    SourceExpression = #39'Altura atual do Form: '#39' + ToStr(Height)'
    ControlExpression = 'Text'
    ...
end
```

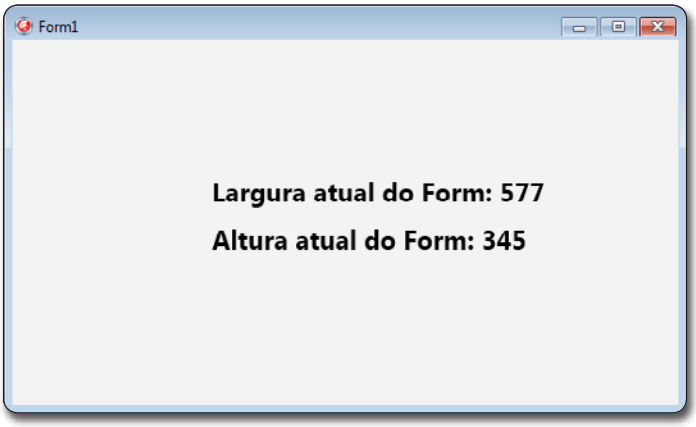


Figura 8. Textos sendo atualizados conforme o Form é redimensionado

A fim de se notificar o mecanismo de expressão, no evento OnResize do Form, invoca-se o método Notify do TBindingsList, conforme já visto no exemplo anterior. A diferença aqui fica por conta do segundo parâmetro, uma vez que estamos lidando com 2 Bindings. É passado para ele então uma String vazia ao invés do nome de uma determinada propriedade. Isso faz com que todos os Bindings sejam notificados ao mesmo tempo. Por fim, o resultado final pode ser visto na **Figura 8**.

Conforme a experiência em lidar com o LiveBindings vai aumentando, fica nítido que a forma de utilizá-lo, seja em aplicações do tipo VCL ou FireMonkey (**Nota do DevMan 2**) é praticamente única, o que acaba por contribuir no aprendizado, reduzindo a curva de aprendizado. Todavia, por se tratar de plataformas distintas, alguns dos recursos disponíveis na VCL podem não serem suportados no FireMonkey e vice-versa. Especificamente no Delphi XE2, o FireMonkey dispõe de um número maior de classes LiveBindings em relação à VCL. Todas estas classes se refletem em componentes relacionados a Link LiveBindings e de controles UI, tais como TBindDBEditLink, TBindDBTextLink, TBindDBCheckLink, TBindDBMemoLink e TBindDBImageLink para associações de dados simples, TBindDBGridLink e TBind DBListLink para associações envolvendo um conjunto de dados e TBindNavigator. Este último, apesar do "Bind" no nome, se difere de todos os outros pelo fato de não ser essencialmente um LiveBindings, mas sim um componente do FireMonkey para se trabalhar com Binding na interface de usuário da aplicação.

Conclusão

Neste artigo, vimos em detalhes o LiveBinding, que é apenas um dos vários recursos incorporados ao Delphi mais recentemente.

Nota do DevMan 2

O FireMonkey é a mais nova plataforma de desenvolvimento da Embarcadero, sendo incorporada ao Delphi em sua versão XE2. Entre suas principais características está seu caráter multi-plataforma (Windows 32-64 bits e Mac OS X), disponibilização de meios para a construção de GUIs modernas através da criação de aplicações que fazem uso da aceleração gráfica de Hardware para suas interfaces. Dessa forma, utiliza-se os recursos disponíveis na API gráfica do Sistema Operacional em questão, tal como Direct2D, OpenGL e GDI+. O FireMonkey, ou simplesmente FMX, traz ainda uma biblioteca de componentes visuais sofisticados, com forte apelo gráfico, suporte à animação e aplicação de estilos, para serem utilizados nos novos tipos de aplicações disponíveis no IDE – FireMonkey HD Application e FireMonkey 3D Application.

Dito isto, se olharmos para a atual documentação oficial do produto, podemos ver o quanto a ferramenta evoluiu e continua evoluindo desde o tradicional Delphi 7. Tudo isso, muito em razão das próprias necessidades de mercado que surgem a todo instante no cenário do desenvolvimento de software. Como de praxe, fica como dica o estudo mais aprofundado do LiveBindings, principalmente com relação ao seu uso em aplicações FireMonkey, que vem ganhando cada vez mais espaço no cenário Delphi.

Agradeço a todos e até a próxima.

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 7 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas. Certified Delphi Developer. Certified Embarcadero Instructor. Blog: prismsilver.blogspot.com

Delphi XE2 – Download Trial
https://downloads.embarcadero.com/free/delphi

Delphi XE2 – Página oficial do produto
http://embarcadero.com/products/delphi

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Generics e Hash Table

Empregue Generics de uma forma simples com as classes do Delphi

Resumo DevMan

De que se trata o artigo:

Este artigo apresenta uma introdução ao recurso de Generics, demonstrando suas características e algumas simples formas de empregá-lo através do Delphi XE2, que traz consigo uma coleção de diversas classes do mesmo, como é o caso da TDictionary, que possibilita o emprego de HashTable nos projetos, uma estrutura constituída por pares de chave-valor.

Em que situação o tema é útil:

Aumentar a produtividade, minimizar possíveis falhas de conversão de tipos em tempo de execução, ter uma maior flexibilidade na manipulação dos dados, dentre outras vantagens que podem ser obtidas ao empregar Generics com o Delphi XE2 e suas classes.

Generics e Hash Table :

Elaborar projetos que possibilitem um aumento da produtividade e redução de falhas é um dos objetivos de qualquer equipe de desenvolvimento. Um recurso presente no Delphi que auxilia na concretização desta meta é o Generics, onde tipos parametrizados podem ser empregados nas mais diversas situações com os mais variados fins. Este artigo apresenta uma breve introdução a este importante recurso, demonstrando algumas das principais classes presentes no Delphi para sua implementação, como é o caso da TDictionary, que por sua vez permite o emprego de HashTable, a estrutura de dados baseada em pares chave-valor, para otimizar pesquisas em grandes volumes de dados.

O desenvolvimento de Software é uma área que conta com uma diversidade de conceitos e recursos que na maioria dos casos possibilitam que aplicações destinadas a um mesmo fim sejam elaboradas das mais variadas formas, cabendo à equipe selecionar as que melhores se adéquem ao projeto em questão. Entretanto, implementar tecnologias que possibilitem a redução de falhas e facilitem o futuro processo de manutenção do Software sempre é uma boa prática. Um recurso interessante que foi adicionado ao Delphi a partir da versão 2009 e trouxe funcionalidades importantes é o Generics. Em síntese, este recurso permite o emprego de tipos parametrizados que possibilitam a elaboração de métodos, classes, registros e interfaces mais flexíveis. Na prática, em um método, por exemplo, um genérico é posteriormente substituído por um tipo, de forma semelhante ao processo que ocorre com os procedimentos e funções convencionais, onde os mesmos recebem os valores através dos argumentos, entretanto neste caso, eles já possuem seus tipos definidos, diferentemente do uso de Generics, que permite a criação de métodos genéricos onde os tipos são resolvidos em tempo de execução.

Normalmente, este recurso também é amplamente empregado para a elaboração de listas compostas por quaisquer tipos de objetos, organizando e auxiliando o processo de acesso aos mesmos, onde no Delphi a classe TList pode ser empregada para este fim.

Outra vantagem que pode ser obtida ao empregar Generics é com relação à redução de código fonte, afinal, muitas vezes nos projetos são criadas várias classes e métodos semelhantes apenas para a manipulação de tipos diferentes, uma tendência que pode ser eliminada com a utilização desta funcionalidade.

Outro recurso que pode ser facilmente empregado com o uso de Generics é o HashTable. Em síntese, HashTable pode ser definido como uma estrutura para o armazenamento de dados com base em uma função Hash, efetuando um mapeamento de chaves para valores. Este recurso permite uma otimização em pesquisas com grandes volumes de dados. No Delphi, a classe responsável por implementar pares baseados em chave-valor é a TDictionary. Vale à pena salientar que uma função Hash, também conhecida como função de dispersão unidirecional, pode receber dados de diversos tamanhos como entrada e retornar um resultado de tamanho fixo de bits, onde a saída recebe a denominação de Hash. Algumas destas funções também podem ser utilizadas para criptografar informações.

Como é possível observar, o emprego de Generics pode trazer diversos benefícios no desenvolvimento das aplicações, tornando-o um recurso interessante para ser adotado nas mais variadas situações.

Generics

Como mencionado, Generics são parâmetros que permitem uma maior flexibilidade no desenvolvimento das classes, métodos e etc. Seu uso no Delphi XE2 pode ser efetuado basicamente através da inclusão das Units System.Generics.Defaults,

System.Generics.Collections e SysUtils na cláusula Uses do formulário em questão, evidentemente dependendo das classes que serão empregadas e das necessidades do projeto. A **Tabela 1** apresenta algumas das principais classes encontradas na Unit System.Generics.Collections.

Como visto, a Unit System.Generics.Collections contém uma diversidade de classes destinadas a implementação de Generics, entretanto a Unit System.Generics.Defaults também é muito importante pois contém interfaces e classes que permitem a implementação de algumas funcionalidades que podem ser utilizadas em conjunto. A **Tabela 2** apresenta algumas destas classes e interfaces presentes em System.Generics.Defaults.

É possível observar que o Delphi oferece a base necessária para que o emprego de Generics seja prático e rápido, permitindo que suas funcionalidades sejam facilmente utilizadas nos projetos.

Tutorial

Generics na prática - TArray

Para dar início a um simples exemplo utilizando Generics com TArray, um novo projeto deve ser criado (File>New>VCL Forms Application). No formulário principal foram adicionados dois objetos do tipo TSpeedButton e um TTreeView. Para que este exemplo possa ser devidamente executado, as Units System.Generics.Collections e System.Generics.Defaults devem ser inclusas na cláusula Uses. Na seção **var** do formulário também foi declarado um array que recebeu a denominação de "ArrayCarros" de acordo com o trecho a seguir:

```
ArrayCarros: TArray<string>;
```

É possível observar que o mesmo é informado como String, mas vale a pena lembrar que poderia ser outro tipo qualquer. Após a declaração do mesmo, um dos botões adicionados deve receber o código fonte para gerar o Array. A **Listagem 1** demonstra o código do evento OnClick do TSpeedButton que foi denominado btGerarArray.

Foi definido um tamanho (5) para o ArrayCarros através do método SetLength (linha 5) e logo em seguida o Array é preenchido com alguns valores (linhas 06 a 10). Posteriormente, o método Sort (linha 11) da classe TArray é chamado para organizar o Array em ordem alfabética, conforme o parâmetro TStringComparer.Ordinal. A classe TStringComparer, presente na Unit System.Generics.Defaults, pode ser empregada para a comparação Case-Sensitive de valores do tipo String. Na linha 12 é efetuada uma limpeza nos itens do objeto TreeView1 através de Items.Clear. Para concluir, é utilizado um laço for para que cada valor presente no ArrayCarros

Classe	Descrição
TList	Possibilita o emprego de uma lista genérica de objetos, tipos primitivos ou registros.
TObjectList	Uma lista genérica apenas para os objetos.
TDictionary	Possibilita a elaboração de uma coleção de pares chave-valor, otimizada para pesquisas.
TObjectDictionary	Uma coleção de pares chave-valor para objetos.
TPair	Uma estrutura para armazenar um par chave-valor.
TArray	Uma classe que reúne diversas funcionalidades para a implementação e manipulação de arrays genéricos.
TEnumerable	Uma classe que é herdada por enumeradores.
TEnumerator	Uma classe que é implementada por enumeradores.
TQueue	Permite a implementação de uma fila genérica.
TObjectQueue	Emprego de fila genérica para objetos.
TStack	Possibilita a implementação de uma pilha genérica (Last In, First Out).
TObjectStack	Emprego de pilha genérica para objetos.

Tabela 1. Algumas classes presentes na Unit System.Generics.Collections

Classe/Interface	Descrição
IComparer	Uma interface genérica para possibilitar a comparação entre dois valores de um mesmo tipo.
TComparer	Classe base para a implementação da interface IComparer.
IEqualityComparer	Uma interface genérica utilizada para verificar se dois valores são idênticos.
TEqualityComparer	Classe base para a implementação da interface IEqualityComparer.
TCustomComparer	Classe base para a comparação que implementa as interfaces IComparer e IEqualityComparer.
THasher	Classe utilizada para gerar o código de Hash de determinado valor.

Tabela 2. Algumas classes e interfaces da Unit System.Generics.Defaults

seja adicionado no objeto TreeView1 através do método Items.Add (linhas 13 e 14).

O próximo passo para este exemplo é implementar uma busca simples neste Array, retornando a posição da palavra pesquisada se a mesma for encontrada. A **Listagem 2** apresenta o código atribuído ao evento OnClick do outro TSpeedButton adicionado, btPesquisarArray.

Listagem 1. Código do evento OnClick do botão btGerarArray

```
01 procedure TFrPrincipal.btGerarArrayClick(Sender: TObject);
02 var
03   StringAux: string;
04 begin
05   SetLength(ArrayCarros, 5);
06   ArrayCarros[0] := 'Corsa';
07   ArrayCarros[1] := 'Linea';
08   ArrayCarros[2] := 'Gol';
09   ArrayCarros[3] := 'Azira';
10   ArrayCarros[4] := 'Civic';
11   TArray.Sort<String>(ArrayCarros, TStringComparer.Ordinal);
12   TreeView1.Items.Clear;
13   for StringAux in ArrayCarros do
14     TreeView1.Items.Add(nil, StringAux);
15 end;
```

Listagem 2. Código do evento OnClick do botão btPesquisarArray

```
01 procedure TFrPrincipal.btPesquisarArrayClick(Sender: TObject);
02 var
03   Palavra: string;
04   Posicao: Integer;
05 begin
06   if InputQuery('Pesquisa', 'Digite o Carro:', Palavra) then
07   begin
08     if TArray.BinarySearch<String>(ArrayCarros, Palavra, Posicao,
       TStringComparer.Ordinal) then
09       ShowMessage(Palavra +' encontrado na posição '+
       IntToStr(Posicao))
10     else
11       ShowMessage('Não encontrado!');
12   end;
13 end;
```

O recurso InputQuery (linha 06) é utilizado para uma rápida obtenção da palavra que deve ser pesquisada e em seguida o método BinarySearch (linha 08) da classe TArray é invocado. Este método efetua a pesquisa em um Array utilizando busca binária, porém, requer que o Array esteja devidamente ordenado. Ele também possui um parâmetro de saída (out) que permite obter a posição (Index) da primeira ocorrência do termo pesquisado, retornando um valor booleano se a pesquisa foi bem sucedida. Para concluir, se a palavra é encontrada, então o usuário recebe a mensagem contendo a posição da mesma no Array (linha 09), caso contrário, ele é alertado que não houve nenhuma ocorrência para o termo (linha 11).

Evidentemente este é um simples exemplo do uso de Generics com TArray, sendo uma alternativa a ser empregada quando houver a necessidade de empregar e pesquisar Arrays nos projetos.

Generics na prática - TList

Outro recurso de Generics que pode ser extensamente utilizado é o TList. Conforme mencionado anteriormente, ele pode conter uma lista genérica composta por qualquer objeto, tipo primitivo e etc. Para dar início a um simples exemplo utilizando este recurso, o mesmo projeto do exemplo anterior pode ser utilizado. Sendo assim, na seção type do formulário principal será declarada uma nova classe que recebeu a denominação de TRevista, que por sua vez conterá propriedades e métodos. A **Listagem 3** apresenta a classe TRevista criada para este exemplo.

Listagem 3. Código da classe TRevista

```
type
  TRevista = class
  private
    FNome: string;
    FEdicao: string;
    FEditora: string;
  published
  public
    property Nome: string read FNome write SetNome;
    property Edicao: string read FEdicao write SetEdicao;
    property Editora: string read FEditora write SetEditora;
  end;
```

Essa classe contém algumas propriedades para armazenar as informações de uma Revista. Após sua adição na seção **type**, basta pressionar simultaneamente as teclas CTRL + SHIFT + C para que o Delphi crie automaticamente a declaração e os códigos para os métodos **write** (set) informados, que serão inseridos na seção **private** da classe de acordo com o trecho a seguir:

```
procedure SetEdicao(const Value: string);
procedure SetEditora(const Value: string);
procedure SetNome(const Value: string);
```

Sendo assim, convém mover tal código para a seção **public**, uma vez que os mesmos devem estar acessíveis externamente.

O passo seguinte é declarar na seção **var** do formulário principal, uma variável do tipo TList<> que conterá objetos instanciados da classe TRevista, para que a mesma esteja acessível por todos os métodos, de acordo com o trecho a seguir:

```
ListaDeRevistas: TList<TRevista>;
```

É possível notar que a lista foi declarada para receber objetos do tipo TRevista através da declaração TList<TRevista>. O próximo passo é criar a instância de TList para que a mesma seja devidamente inicializada. Sendo assim, o evento OnCreate do formulário principal recebeu o trecho de código a seguir:

```
ListaDeRevistas := TList<TRevista>.Create;
```

Isto faz com que a lista seja criada no momento da geração do formulário. A etapa seguinte para este exemplo é popular a lista

com algumas informações para que a mesma possa ser posteriormente acessada e algumas de suas funcionalidades apresentadas. Sendo assim, um novo botão do tipo TSpeedButton foi adicionado ao formulário. A **Listagem 4** demonstra o código empregado no evento OnClick do botão inserido para popular a lista de uma forma prática.

Listagem 4. Código do evento OnClick do botão btPopularLista

```
01 procedure TFrPrincipal.btPopularListaClick(Sender: TObject);
02 var
03   Revista: TRevista;
04   TotalDeRevistas, i: Integer;
05 begin
06   Revista := TRevista.Create;
07   with Revista do
08   begin
09     SetNome('Clube Delphi');
10     SetEdicao('146');
11     SetEditora('DevMedia');
12   end;
13   ListaDeRevistas.Add(Revista);
14   Revista := TRevista.Create;
15   with Revista do
16   begin
17     SetNome('Java Magazine');
18     SetEdicao('109');
19     SetEditora('DevMedia');
20   end;
21   ListaDeRevistas.Add(Revista);
22   Revista := TRevista.Create;
23   with Revista do
24   begin
25     SetNome('.Net Magazine');
26     SetEdicao('100');
27     SetEditora('DevMedia');
28   end;
29   ListaDeRevistas.Add(Revista);
30   Revista := TRevista.Create;
31   with Revista do
32   begin
33     SetNome('Mobile Magazine');
34     SetEdicao('45');
35     SetEditora('DevMedia');
36   end;
37   ListaDeRevistas.Add(Revista);
38   TotalDeRevistas := ListaDeRevistas.Count;
39   TreeView1.Items.Clear;
40   for i := 0 to TotalDeRevistas -1 do
41     TreeView1.Items.Add(nil, ListaDeRevistas.Items[i].Nome);
42 end;
```

Neste trecho de código da **Listagem 4** é definida uma variável do tipo TRevista (linha 03) para auxiliar no processo de criação (TRevista.Create – linha 06) e adição das revistas na lista (ListaDeRevistas.Add – linha 15 a 37). Após definidas as propriedades das quatro revistas, o total de objetos presentes na lista é atribuído à variável TotalDeRevistas através da propriedade Count do objeto ListaDeRevistas (TList – linha 38). Em seguida, se existirem itens no objeto TreeView1, os mesmos são eliminados através do método Items.Clear (linha 39). Para concluir, é efetuado um laço **for** para varrer a lista e popular o objeto TreeView1 através do método Items.Add (linhas 40 e 41), de acordo com a propriedade Nome dos objetos que compõem a lista. Neste momento é possível notar uma das diferenças ao empregar Generics nos projetos, as propriedades que pertencem a um objeto instanciado da classe TRevista podem ser acessados diretamente, sem que exista a necessidade de utilizar TypeCasting (ver **Nota do DevMan 1**). Sem o uso de Generics, o código empregado seria semelhante, porém haveriam algumas divergências conforme apresentado na **Listagem 5**.

Listagem 5. Adição de objetos sem o uso de Generics

```
01 ...
02 var
03   Lista: TList;
04 begin
05   Lista := TList.Create;
06 ...
07   Lista.Add(Revista);
08 ...
09   for i := 0 to TotalDeRevistas -1 do
10     TreeView1.Items.Add(nil, TRevista(Lista.Items[i]).Nome);
11 ...
```

Nota do DevMan 1

O TypeCasting é um recurso muito importante presente em muitas linguagens de programação e que permite a conversão/mudança de tipos. No Delphi, o TypeCasting pode ser efetuado basicamente de duas formas, utilizando "Objeto as TClasse" ou "TClasse(Objeto)". Existe uma diferença entre ambas. Na primeira, é automaticamente efetuada uma verificação para testar se a chamada é válida em tempo de execução, facilitando o processo de identificação de possíveis falhas caso ocorram, tais como objetos que não são compatíveis com a classe informada. Já na segunda, não ocorre uma verificação, onde o objeto em questão é "forçado" a ser considerado como originado da classe informada, o que poderia ocasionar erros inespecíficos como Access Violation quando os mesmos são incompatíveis, sendo assim, tal recurso deve ser utilizado com muito cuidado e apenas quando existe a certeza de que o objeto em questão realmente é condizente com aquela classe.

É possível perceber neste exemplo, que não emprega Generics, a necessidade de utilizar TypeCasting para que a propriedade "Nome" do objeto em questão fosse acessada (TRevista(Lista.Items[i]).Nome). Sendo assim, em aplicações mais complexas que utilizassem um número maior de classes com características distintas, erros poderiam surgir em tempo de execução, que evidentemente podem ser contornados com o emprego de uma verificação antes do TypeCasting, testando se o objeto em questão é do tipo esperado, o que implicaria em mais código fonte sendo empregado toda vez que o objeto precisasse ser acessado na lista, algo que não necessita ser implementado quando o recurso de Generics é adotado.

Para concluir o exemplo utilizando TList com Generics, foi inserido o método Free para a lista (ListaDeRevistas.Free) no momento em que o formulário é destruído (OnDestroy) para sua devida liberação da memória.

Como visto, Generics permitem que erros durante o tempo de execução sejam evitados, uma vez que caso houvesse alguma falha por causa de tipo, a mesma seria exibida no momento da compilação.

Generics na prática - TDictionary e HashTable

Outro recurso presente em Genercis muito útil é o TDictionary. Ele permite o armazenamento de determinado valor utilizando como base uma chave, em outras palavras, possibilita a criação de uma lista baseada em pares constituídos por chave-valor, onde ambos podem ser de qualquer tipo, de forma semelhante ao recurso HashTable em C#. Ele é otimizado para possibilitar buscas mais rápidas, também se aparentando a uma matriz. Apesar de ser simples de empregar, o TDictionary permite a criação de estruturas de dados complexas com grandes volumes de dados.

Para iniciar um exemplo prático deste recurso, o mesmo projeto que vem sendo empregado pode ser utilizado, bem como a classe TRevista que foi criada no exemplo anterior. Sendo assim, a seção **var** do formulário principal recebeu a declaração do TDictionary de acordo como trecho a seguir:

```
Dicionario: TDictionary<string, TRevista>;
```

Como é possível notar, foram adotados pares constituídos por string-TRevista para o objeto TDictionary, lembrando que qualquer tipo poderia ser adotado.

Para dar sequência, antes de ser utilizado, o objeto do tipo TDictionary necessita ser criado. Da mesma forma como o objeto TList do exemplo anterior foi definido no evento OnCreate do formulário, o TDictionary também pode ser gerado em tal momento. Sendo assim, tal evento recebeu a adição do código a seguir:

```
Dicionario := TDictionary<string, TRevista>.Create;
```

Concluída a devida inicialização do objeto TDictionary, a próxima etapa é preenchê-lo com os valores para que algumas funcionalidades do mesmo possam ser demonstradas. Para tanto, foi adicionado mais um botão do tipo TSpeedButton no formulário. A **Listagem 6** apresenta o código atribuído ao evento OnClick do último botão adicionado, btPopularDicionario.

É possível notar que o código é semelhante ao empregado na **Listagem 4** para popular o objeto TList. Entretanto, o método Add (linha 12) do objeto TDictionary recebe dois argumentos, diferentemente do objeto TList que recebe apenas um. O primeiro argumento para o método é a chave que será utilizada, neste caso uma string, e o segundo, o valor que neste exemplo é uma instância da classe TRevista.

Objetos do tipo TDictionary permitem uma fácil verificação antes da adição de um novo par, uma vez que não devem existir chaves duplicadas, através do método ContainsKey. Se desejasse efetuar tal validação, o desenvolvedor poderia empregar um código semelhante ao apresentado a seguir:

```
if Dicionario.ContainsKey('Delphi') then
  ShowMessage('Esta revista já foi inserida!')
else
  Dicionario.Add('Delphi', Revista);
```

Listagem 6. Código do evento OnClick do botão btPopularDicionario

```
01 procedure TFrPrincipal.btPopularDicionarioClick(Sender: TObject);
02 var
03   Revista: TRevista;
04 begin
05   Revista := TRevista.Create;
06   with Revista do
07   begin
08     SetNome('Clube Delphi');
09     SetEdicao('146');
10     SetEditora('DevMedia');
11   end;
12   Dicionario.Add('Delphi', Revista);
13   Revista := TRevista.Create;
14   with Revista do
15   begin
16     SetNome('Java Magazine');
17     SetEdicao('109');
18     SetEditora('DevMedia');
19   end;
20   Dicionario.Add('Java', Revista);
21   Revista := TRevista.Create;
22   with Revista do
23   begin
24     SetNome('.Net Magazine');
25     SetEdicao('100');
26     SetEditora('DevMedia');
27   end;
28   Dicionario.Add('.Net', Revista);
29   Revista := TRevista.Create;
30   with Revista do
31   begin
32     SetNome('Mobile Magazine');
33     SetEdicao('45');
34     SetEditora('DevMedia');
35   end;
36   Dicionario.Add('Mobile', Revista);
37 end;
```

Tal verificação é extremamente prática e necessária, pois, ocorrerá uma exceção ao tentar inserir novamente uma chave que já exista no objeto TDictionary. Para dar sequência ao exemplo, foi adicionado um novo TSpeedButton com o intuito de implementar uma simples pesquisa para um objeto TDictionary. A **Listagem 7** demonstra o código do evento OnClick do último botão inserido, btPesquisarDicionario.

Também é utilizado o recurso InputQuery (linha 06) para a obtenção da revista que deve ser pesquisada. Posteriormente, o método TryGetValue (linha 07) do objeto TDictionary é empregado, uma vez que o mesmo efetua uma busca na tentativa de encontrar a informação solicitada. O primeiro parâmetro é a chave pela qual o mesmo deve buscar e o segundo é uma saída (out) para possibilitar a obtenção do devido valor de acordo com a chave, evidentemente se tal chave for encontrada, onde neste caso é obtido um objeto do tipo TRevista para a variável RevistaAux, lembrando que tal método retorna um valor booleano de acordo com o sucesso da pesquisa. Em seguida, se o mesmo foi localizado, então é exibida uma mensagem apresentando as propriedades do objeto encontrado (linha 08) e caso contrário, o usuário é alertado que não foi possível encontrar tal informação (linha 11).

O próximo passo para este exemplo será elaborar um código para exibir todas as informações contidas no objeto TDictionary através do TreeView1, presente no formulário. Um novo botão foi adicionado ao formulário. A **Listagem 8** apresenta o código do evento OnClick para o botão inserido, btExibirRevistas.

São declaradas algumas variáveis para auxiliar no processo de busca no dicionário e apresentação das informações através do objeto TreeView1 (linhas 03 a 05). Após o início do bloco, todos os itens do TreeView1 são removidos através do método Clear (linha 07). Posteriormente, é executado um trecho de código para cada chave presente no objeto TDictionary através do laço for (linha 08). As chaves podem ser facilmente obtidas através da propriedade Keys de um objeto do tipo TDictionary. De forma semelhante, os valores também podem ser adquiridos diretamente com um laço for através da propriedade Values. Em seguida, o objeto Node (TTreeNode), que representa um nó da árvore (TTreeView), recebe o nó retornado no momento da adição de um novo item no objeto TreeView1, que por sua vez recebeu como sua descrição, o valor da chave que é string. Em seguida, são adicionados filhos a este nó contendo as informações do objeto revista presente no dicionário através do método AddChild (linhas 13 a 15). Mais uma vez não existe a necessidade de utilizar TypeCasting para acessar as informações da revista, já que o recurso de Generics é empregado. Também é possível notar que, para obter a revista referente à chave atual, basta utilizar Dicionario.Items[Chave]. Para concluir, é utilizado um for para expandir todos os elementos do objeto TreeView1 através da propriedade Expand de cada item definida como True (linhas 18 e 19).

Listagem 7. Código do evento OnClick do btPesquisarDicionario

```
01 procedure TFrPrincipal.btPesquisarDicionarioClick(Sender: TObject);
02 var
03   Palavra: string;
04   RevistaAux: TRevista;
05 begin
06   if InputQuery('Pesquisa', 'Digite o Carro:', Palavra) then
07     if (Dicionario.TryGetValue(Palavra, RevistaAux)) then
08       ShowMessage('Nome: '+ RevistaAux.Nome +#13+
             'Edição: '+ RevistaAux.Edicao +#13+
             'Editora: '+ RevistaAux.Editora)
09     else
10       ShowMessage('Revista '+ Palavra +' não encontrada!');
11 end;
```

Listagem 8. Código do evento OnClick do btExibirRevistas

```
01 procedure TFrPrincipal.btExibirRevistasClick(Sender: TObject);
02 var
03   Chave: string;
04   Node: TTreeNode;
05   i: Integer;
06 begin
07   TreeView1.Items.Clear;
08   for Chave in Dicionario.Keys do
09   begin
10     Node := TreeView1.Items.Add(nil, Chave);
11     with TreeView1.Items do
12     begin
13       AddChild(Node, Dicionario.Items[Chave].Nome);
14       AddChild(Node, Dicionario.Items[Chave].Edicao);
15       AddChild(Node, Dicionario.Items[Chave].Editora);
16     end;
17   end;
18   for i := TreeView1.Items.Count -1 downto 0 do
19     TreeView1.Items.Item[i].Expand(True);
20 end;
```

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software. Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviços); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DevMedia

Como foi possível observar neste exemplo, utilizar HashTable é um processo extremamente simples com Generics, lembrando que qualquer tipo poderia ser adotado para a chave ou para o valor. A **Figura 1** apresenta a aplicação em execução.

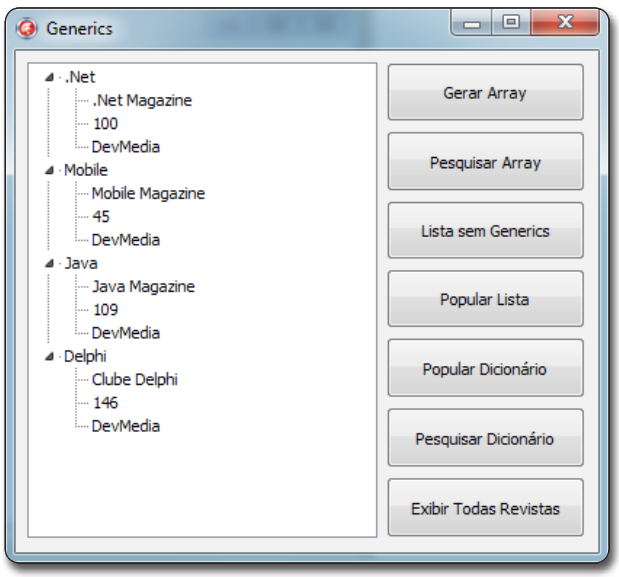


Figura 1. Aplicação com Generics em execução

Conclusão

Generics é um recurso muito interessante para ser empregado nos projetos, afinal, permite uma maior flexibilidade, aumenta a produtividade, dentre outras características. De início, ele pode até assustar um pouco em meio à forma que é empregado, mas é algo simples de ser utilizado no cotidiano. Além do mais, ao utilizar este recurso, eventuais erros que poderiam ocorrer em tempo de execução durante os processos de conversão de tipos são eliminados.

Quanto ao Delphi, suas versões mais recentes oferecem diversas classes que podem ser empregadas nos projetos para a implementação deste recurso. Entretanto, este artigo apresentou um simples exemplo da utilização de Generics com os recursos já disponibilizados, porém vale à pena lembrar que suas funcionalidades vão muito além do que foi apresentado, trazendo diversas possibilidades para o desenvolvedor que deseja empregá-lo em seus projetos.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Informações sobre Generics no Delphi XE2
http://docwiki.embarcadero.com/RADStudio/XE2/en/Overview_of_Generics

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Dificuldades em encontrar conteúdo sobre .Net para iniciantes?

A DevMedia simplifica pra você!

easy .net magazine

Easy .Net Magazine:

A primeira revista digital feita especialmente para quem está dando os primeiros passos em .Net!

Você encontrará exemplos e modelos fáceis e práticos, noções introdutórias e conceitos básicos.

Conheça as vantagens:

- Vídeos e apresentações "embutidos" nas páginas;
- Ilustrações, diagramas e modelos apresentados em PowerPoint;
- Entrevistas em vídeo;
- Códigos-fonte que podem ser copiados com CTRL+C/CTRL+V.

A Easy.Net não é feita em PDF ou Flash!
Ela utiliza um aplicativo 100% online ou no DevPlayer.

DevMedia group

Acesse agora a edição Nº 1 totalmente grátis e comprove

www.devmedia.com.br/easynet

Easy.Net Magazine, seu curso inicial de .Net!

Aplicação de Backup/ Restore no Firebird

Aprenda como criar uma interface gráfica para seu backup

O Firebird, desde seu nascimento, conta com alguns utilitários de linha de comando que podem (e devem) ser utilizados para manutenção e obtenção de estatísticas do banco de dados. Através destes, é possível realizar tarefas como Backup/Restore, verificações de índices, monitoramento de transações, correção do banco de dados, reindexação, limpeza do coletor de lixo (Garbage Collector – ver **Nota do DevMan 1**), além de outras tantas tarefas, como o Trace do banco (ver **Nota do DevMan 2**).

Embora exista essa grande quantidade de utilitários, todos são baseados em uma interface de comandos o que claramente pode proporcionar alguns pontos um tanto quanto desagradáveis. Isto porque a interface de comandos geralmente não é amigável, o que dificulta a visualização das informações, análise dos resultados e é claro, requer um conhecimento/domínio maior sobre os comandos, parâmetros e etc.

Nota do DevMan 1

Garbage Collector: O Garbage Collector é um recurso bastante conhecido em programação. Neste caso, ele é responsável por fazer o gerenciamento de objetos que não foram destruídos, encarregando de gerenciá-los na memória e evitando assim os famosos Memory Leaks. Assim como no conceito da programação, o Garbage Collector está presente em bancos de dados. No Firebird não é diferente. Por padrão, o Firebird automaticamente gerencia os registros previamente deletados do banco de dados, de qualquer modo, realizar os processos de Backup/Restore do banco de dados realizam uma completa limpeza no banco de dados, fazendo com que o banco inclusive reduza drasticamente em algumas situações.

Nota do DevMan 2

Trace: O recurso de Trace nada mais é do que o monitoramento completo das atividades do servidor de banco de dados. Neste caso é possível realizar uma completa auditoria utilizando esta ferramenta, mapeando os detalhes de cada conexão ativa.

Resumo DevMan

De que se trata o artigo:

O artigo trata de demonstrar a criação de uma interface gráfica para realização de operações de Backup/Restore utilizando o Firebird e o utilitário de linha de comando Gbak, tornando a maneira de realizar estas tarefas muito mais amigável.

Em que situação o tema é útil:

O tema é útil quando se deseja implementar aplicações personalizadas para automatização de tarefas cotidianas, fazendo com que o processo de realizar backup/restore seja mais simples, podendo ser realizado por pessoas sem um alto nível de conhecimento técnico.

Aplicação de Backup/Restore no Firebird:

Tarefas como Backup/Restore são comuns. Principalmente no caso do Backup, já que é recomendado como uma forma de prevenção de possíveis problemas, tornando assim o seu ambiente menos suscetível a erros. Como já abordado em diversos artigos meus, a criação de aplicativos para a realização de tarefas rotineiras pode poupar grandes dores de cabeça. Embora através de uma busca rápida pelo Google seja possível obter inúmeros aplicativos como este, nada melhor do que criar sua própria ferramenta e adaptá-la conforme a necessidade. Sendo assim, neste artigo veremos como criar uma ferramenta para automatizar estes processos.

A maior justificativa para a criação deste tipo de aplicação é fazer com que o próprio usuário ou um suporte técnico esteja apto a fazer o backup ou restauração do banco de dados de forma simples e ágil, o que traz como benefício uma redução no tempo gasto para este tipo de tarefa. Vale lembrar que este tipo de recurso pode ser facilmente incorporado ao seu sistema, além de permitir adicionar uma gama grande de outros recursos como backup agendado, compactação dos arquivos e envio por FTP por exemplo.

O principal utilitário usado para a elaboração do nosso aplicativo será o gbak. O gbak está disponível desde a primeira versão do Firebird (geralmente dentro da pasta bin da instalação padrão).

Através dele é possível utilizar passagem de parâmetros para definir as melhores maneiras de realizar tanto o Backup quanto o Restore do banco de dados, possuindo uma grande variedade de parâmetros configuráveis. O processo de Backup é dado basicamente pela chamada do gbak passando como parâmetro o local do banco de dados, usuário e senha e o local dos arquivos a serem gerados, que possuirão a extensão .fbk. O Restore é realizado praticamente do mesmo modo, porém, invertendo os parâmetros de .fbk e caminho do banco. Veremos isto no decorrer do artigo e abordaremos um pouco sobre os tipos de parâmetros, bem como suas respectivas funcionalidades.

Definindo a aplicação

Como anteriormente dito, o Firebird embora possua inúmeras ferramentas para auxílio de tarefas relativas ao banco de dados, estas não utilizam interfaces gráficas, sendo utilizadas e manipuladas principalmente através de linhas de comando (prompt de comandos). Neste caso, criaremos uma interface gráfica que realizará algumas das tarefas disponíveis vinha linha de comandos. Por se tratar de uma aplicação de baixa complexidade, não utilizaremos bancos de dados para realizar a persistência das configurações, que neste caso serão feitas por meio de arquivos XML. Vale apena aqui fazer uma breve comparação dos tipos de arquivo normalmente utilizados em criações de aplicações, o que pode justificar melhor a escolha do formato .xml.

Geralmente, quando há a necessidade de criar arquivos de configurações, as maneiras mais comuns são:

- Arquivos de extensão .ini
- Arquivos de registro .reg
- Arquivos xml/binários do ClientDataSet

Arquivos de extensão .ini são demasiadamente utilizados em aplicações por ser facilmente criados, configurados e manipulados por qualquer pessoa. Comumente suas configurações são separadas em forma de sessões/configurações da sessão.

Exemplo:

```
[BANCO DE DADOS]
Caminho = C:\DB.fdb
```

Uma das grandes vantagens como se pode notar, é sua simplicidade. Já como desvantagem, observa-se que este tipo de arquivo é muito suscetível a erros. O Delphi disponibiliza a unit IniFiles para a total manipulação deste tipo de arquivo.

Por outro lado, arquivos de registro (.reg) são utilizados para armazenamento de configurações no registro do Windows. Este é um detalhe importante, pois este tipo de implementação apenas é funcional nesta plataforma. Seu manuseamento é realizado pelo Delphi através da unit Registry.

Por último, temos os arquivos de configuração xml/binário. As vantagens destes tipos de arquivo tratam da facilidade de manipulação e utilização como fonte de dados, tornando assim a forma de manuseio semelhante às fontes de dados. Tanto o arquivo XML quanto o binário são extremamente leves de serem utilizados (mesmo que o arquivo binário neste quesito leve vantagem). Estes arquivos são gerados facilmente pelo ClientDataSet através do método SaveToFile, configurando os parâmetros dfXML e dfBinary ou diretamente na extensão do arquivo a ser gerado. Outros fatores a serem considerados são a facilidade para a criação dos campos que farão parte das configurações utilizadas e também a redução considerável dos erros, já que a manipulação de arquivos XML podem ser realizadas por qualquer tipo de editor. Esta é uma vantagem em relação ao arquivo binário, pois este possui uma dificuldade de manipulação maior, o que de certa forma pode proporcionar um aumento das questões de segurança.

Explanações realizadas, podemos iniciar a definição do funcionamento da nossa aplicação. O objetivo aqui é armazenar dois tipos de configurações, deste modo, teremos as definições gerais para backup (relacionadas aos parâmetros para backup e caminho do gbak) e as demais, que tratarão de armazenar os possíveis bancos de dados cadastrados para backup. Desta forma, é possível implementar facilmente recursos de backups agendados ou mesmo envio de arquivos de backup para ftp ou e-mail (dependendo do tamanho do arquivo). Sendo assim, mãos a obra.

Tutorial

Criando a aplicação de exemplo

Para iniciar nosso exemplo, criaremos uma nova aplicação do tipo VCL Forms (File>New>VCL Forms Application – Delphi). Salve o seu projeto com o nome de AppBackup. Como já mencionado, trabalharemos com duas estruturas de configurações (configurações de backup e de bancos). Sendo assim, utilizaremos para isso dois ClientDataSets (cdsConfiguracoesApp e cdsConfiguracoesDB). Para a manipulação dos arquivos, utilizaremos duas variáveis criadas especificamente para armazenar os respectivos caminhos. Como de costume, são declaradas no escopo private conforme a **Listagem 1**.

Listagem 1. Definição das variáveis de configuração

```
01 private
02   FPathApp,
03   FPathDB : String;
04             { Private declarations }
05 public
06             { Public declarations }
07 end;
```

Após a definição das variáveis, o próximo passo faz com que sejam atribuídos os diretórios que serão utilizados para a manipulação dos arquivos, para isso é definido o evento OnCreate do formulário com o código da **Listagem 2**.

Nota-se que foi utilizada a função ExtractFilePath que tem por objetivo retornar o caminho completo de um executável, que no caso é o próprio aplicativo. Estes caminhos são concatenados com os nomes dos arquivos de configuração (configApp.xml e configDB.xml). A partir daí é possível manipulá-los de forma simples. O próximo passo é carregar o arquivo (caso o mesmo exista) ou criá-lo, caso contrário. Isto é feito no evento OnShow do formulário, definido na **Listagem 3**.

Na linha 03 é verificada a existência do arquivo de configuração geral (FPathApp), caso o mesmo exista, automaticamente será carregado para o ClientDataSet através do método LoadFromFile (linha 04), caso contrário é utilizado o procedimento CreateDataSet (linha 06), responsável por preparar o ClientDataSet para inserção dos dados. As linhas 07 a 10 tem a mesma finalidade, porém, com o outro arquivo de definições.

Aplicadas as configurações referente ao carregamento dos arquivos, faremos as devidas implementações dos métodos responsáveis por armazenar as alterações em disco. Para isso, utilizamos os eventos AfterPost dos ClientDataSets com o método SaveToFile, passando como parâmetro o caminho dos respectivos arquivos (**Listagem 4**).

Listagem 2. Evento OnCreate do formulário

```
01 procedure TForm1.FormCreate(Sender: TObject);
02 begin
03   FPathApp := ExtractFilePath(Application.ExeName) + 'configApp.xml';
04   FPathDB  := ExtractFilePath(Application.ExeName) + 'configDB.xml';
05 end;
```

Listagem 3. Evento OnShow do formulário

```
01 procedure TForm1.FormShow(Sender: TObject);
02 begin
03   if FileExists(FPathApp) then
04     cdsConfiguracoesApp.LoadFromFile(FPathApp)
05   else
06     cdsConfiguracoesApp.CreateDataSet;
07   if FileExists(FPathDB) then
08     cdsConfiguracoesDB.LoadFromFile(FPathDB)
09   else
10     cdsConfiguracoesDB.CreateDataSet;
11 end;
```

Listagem 4. Eventos OnAfterPost dos ClientDataSets cdsConfiguracoesDB e cdsConfiguracoesApp respectivamente

```
01 procedure TForm1.cdsConfiguracoesDBAfterPost(DataSet: TDataSet);
02 begin
03   cdsConfiguracoesDB.SaveToFile(FPathDB);
04 end;

05 procedure TForm1.cdsConfiguracoesAppAfterPost(DataSet: TDataSet);
06 begin
07   cdsConfiguracoesApp.SaveToFile(FPathApp);
08 end;
```

A manipulação dos arquivos está concluída, porém, ainda não foram definidos os campos. Sendo assim crie os campos nos ClientDataSets conforme as **Tabelas 1** e **2**.

Nome do campo	Tipo	Tamanho
CAMINHO_GBAK	String	250
DESATIVA_INDICE	Boolean	-
IGNORA_CHECKSUM	Boolean	-
IGNORA_TRANSACOES	Boolean	-
GERA_METADATA	Boolean	-
BACKUP_TRANSPORTAVEL	Boolean	-

Tabela 1. Campos do cdsConfiguracoesApp

Nome do campo	Tipo	Tamanho
NOME_DB	String	250
CAMINHO_DB	String	250
DESTINO_BACKUP	String	250
USUARIO	String	250
SENHA	String	250

Tabela 2. Campos do cdsConfiguracoesDB

Nota

Ao trabalhar com DataSets de memória, nunca utilize o método ApplyUpdates. Lembre-se que este método só é utilizado para persistência em banco de dados.

Terminadas as configurações passaremos para a definição do Layout da aplicação. Para isso faremos a divisão do Layout em três etapas. A parte do topo do formulário deve ser definida com um Panel com Align AlTop, contendo dentro de si um Label, um DBEdit, um SpeedButton e um DBNavigator. Observe o resultado na **Figura 1.**

Repare na **Figura 1** que o DBNavigator teve habilitada a propriedade VisibleButtons apenas os botões nbDelete e nbPost. Isto porque deverá haver apenas uma configuração para o aplicativo, porém, pode haver N configurações de banco de dados. Adicione neste momento dois DataSources (dsrConfiguracoes App e dsrConfiguracoesDB) ligando-os respectivamente aos ClientDataSets. O DBEdit e o DBNavigator da **Figura 1** deverão ser conectados ao DataSource dsrConfiguracoesApp. No DBEdit deverá ser informado o campo "CAMINHO_GBAK". Após isso, faremos a implementação do botão. O código do evento OnClick pode ser visualizado na **Listagem 5**.

Note que na linha 06 é criado um objeto TOpenDialog. Este será utilizado para a busca do arquivo gbak no diretório padrão do Firebird (geralmente a pasta bin). Na linha 07 é verificado se há 1 configuração, neste caso o registro é editado (linha 08), caso contrário é realizada uma operação de inserção (linha 10). Na linha 11 é realizada a operação de filtro no objeto oDlg. Isto fará com que sejam apenas filtrados executáveis, exibindo na descrição que deverá ser escolhido um aplicativo gbak. Finalmente na linha 12, se houver uma seleção válida de arquivo, automaticamente será atribuída ao field CAMINHO_GBAK. Por fim o objeto é destruído e liberado da memória (linha 15).

Listagem 5. Evento OnClick do botão de busca do executável GBAK

```
01 procedure TForm1.spbBuscaGBAKClick(Sender: TObject);
02 var
03   oDlg : TOpenDialog;
04 begin
05   try
06     oDlg := TOpenDialog.Create(nil);
07     if cdsConfiguracoesApp.RecordCount = 1 then
08       cdsConfiguracoesApp.Edit
09     else
10       cdsConfiguracoesApp.Append;
11     oDlg.Filter := 'Arquivo do tipo gbak (*.exe)|*.exe';
12     if oDlg.Execute then
13       cdsConfiguracoesAppCAMINHO_GBAK.AsString := oDlg.FileName;
14   finally
15     FreeAndNil(oDlg);
16   end;
17 end;
```

Listagem 6. Evento OnNewRecord do cdsConfiguracoes

```
01 procedure TForm1.cdsConfiguracoesAppNewRecord(DataSet: TDataSet);
02 begin
03   if cdsConfiguracoesApp.RecordCount >= 1 then
04   begin
05     cdsConfiguracoesApp.Cancel;
06     Application.MessageBox('Atenção, só pode haver uma configuração por
07       aplicativo!','Aviso!', MB_ICONWARNING + MB_OK);
08     Exit;
09   end;
10   cdsConfiguracoesAppDESATIVA_INDICE.AsBoolean          := False;
11   cdsConfiguracoesAppIGNORA_CHECKSUM.AsBoolean          := False;
12   cdsConfiguracoesAppIGNORA_TRANSACOES.AsBoolean        := False;
13   cdsConfiguracoesAppGERA_METADATA.AsBoolean            := False;
14   cdsConfiguracoesAppBACKUP_TRANSPORTAVEL.AsBoolean := False;
15 end;
```

Listagem 7. Função CanDoBackup

```
01 function TForm1.CanDoBackup: Boolean;
02 begin
03   Result :=
     ( not cdsConfiguracoesAppCAMINHO_GBAK.IsNull and
       not cdsConfiguracoesDBNOME_DB.IsNull and
       not cdsConfiguracoesDBCAMINHO_DB.IsNull and
       not cdsConfiguracoesDBDESTINO_BACKUP.IsNull and
       not cdsConfiguracoesDBUSUARIO.IsNull and
       not cdsConfiguracoesDBSENHA.IsNull );
04 end;
```

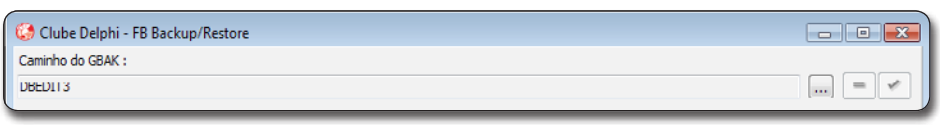


Figura 1. Componentes do topo do formulário principal

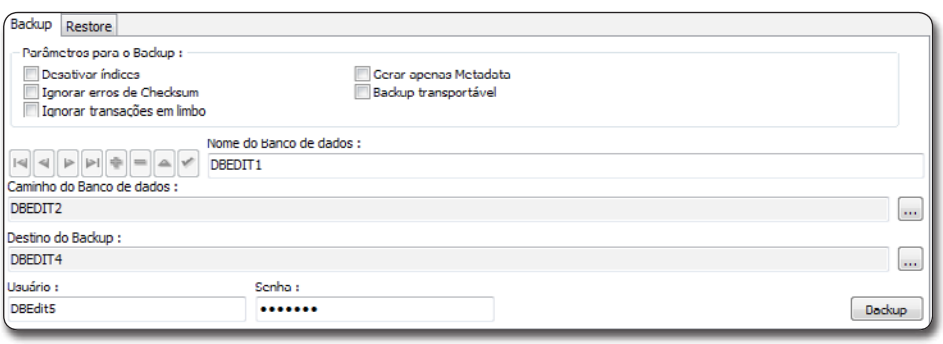


Figura 2. Layout central

Antes de prosseguir na disposição do Layout, faremos com que sejam realizadas as atribuições validações iniciais no cdsConfiguracoesApp. Para isso utilizaremos o evento OnNewRecord, conforme mostrado na **Listagem 6**.

Essa inicia com uma checagem do RecordCount do cdsConfiguracoesApp (linha 03). Caso haja algum registro, o DataSet é cancelado e é exibida uma mensagem (linhas 05 a 06). Isto garante que exista apenas uma configuração válida no ClientDataSet. As demais linhas 10 a 14 apenas inicializam os campos com o valor booleano False.

Continuando com a definição do Layout, a parte central da aplicação deverá ser composta por 1 PageControl definido com alinhamento AlClient e duas abas. Dentro da aba de Backup (primeira aba) deverá ser adicionado 1 GroupBox contendo 5 DBCheckBoxes. Todos estes devem apontar para o DataSource dsrConfiguracoes App. A seguir, deve ser adicionado ao formulário 5 DBEdits, 5 Labels, 1 DBNavigator e 3 SpeedButtons. Os componentes DataAware citados deverão apontar para o DataSource dsrConfiguracoesDB. Como especificaremos primeiramente a parte responsável por realizar os backups, trataremos então esta aba inicialmente. O layout proposto encontra-se ilustrado na **Figura 2**.

Criaremos uma função utilizada para habilitar o botão de backup conforme o preenchimento dos campos obrigatórios. Para isso codifique a função CanDoBackup, conforme mostrado na **Listagem 7**.

Nota

Os detalhes completos de configurações entre os campos e os componentes serão omitidos, no entanto o exemplo completo encontra-se disponível para download no site da DevMedia.

A **Listagem 7** faz com que o resultado da função CanDoBackup seja True caso todos os campos necessários para a realização do backup via gbak sejam preenchidos corretamente. Veremos sua utilização mais adiante.

A próxima etapa é definir uma outra função que retornará os parâmetros pré-configurados conforme a seleção dos campos em tela. **Observe a função getParametros-Backup** na **Listagem 8**.

Na **Listagem 8** os parâmetros são obtidos e concatenados conforme a seleção dos DBCheckBoxes da aplicação. Note nas linhas ímpares a partir da 05 a presença de parâmetros definidos no formato "–sigla". A explanação de cada um dos parâmetros utilizados na função e outros que vão além se encontram descritos na **Tabela 3.**

Retomando a parte de implementação, faremos a codificação do primeiro botão de pesquisa visto na **Figura 2.** Este botão será responsável por localizar os dados do banco de dados de origem (na qual resultará em um arquivo de backup). A **Listagem 9** ilustra o seu evento OnClick.

Listagem 8. Função getParametrosBackup

```
01 function TForm1.getParametrosBackup : String;
02 begin
03  Result := '';
04  if cdsConfiguracoesAppDESATIVA_INDICE.AsBoolean then
05   Result := ' -i ';
06  if cdsConfiguracoesAppIGNORA_CHECKSUM.AsBoolean then
07   Result := Result + ' -ig ';
08  if cdsConfiguracoesAppIGNORA_TRANSACOES.AsBoolean then
09   Result := Result + ' -l ';
10  if cdsConfiguracoesAppGERA_METADATA.AsBoolean then
11   Result := Result + ' -m ';
12  if cdsConfiguracoesAppBACKUP_TRANSPORTAVEL.AsBoolean then
13   Result := Result + ' -t ';
14 end;
```

Listagem 9. Evento onClick do botão de pesquisa de bancos de dados.

```
01 procedure TForm1.spbBancoClick(Sender: TObject);
02 var
03  oDlg : TOpenDialog;
04 begin
05  if not (cdsConfiguracoesDB.State in [dsEdit, dsInsert]) then
06   Exit;
07  oDlg := TSaveDialog.Create(nil);
08  try
09   oDlg.Filter := 'Arquivos de banco de dados (.fdb)|*.fdb';
10   if oDlg.Execute then
11    cdsConfiguracoesDBCAMINHO_DB.AsString := oDlg.FileName;
12  finally
13   FreeAndNil(oDlg);
14  end;
15 end;
```

Como se pode notar, a **Listagem 9** possui uma implementação semelhante a vista anteriormente na **Listagem 5.** Na linha 05 é verificado se o estado do DataSet é de inserção ou edição. Caso não seja, automaticamente é chamado o comando Exit (linha 06). Caso estejam em estado de edição ou inserção, automaticamente o código é executado filtrando bancos de dados (arquivos do tipo .fdb – linhas 09 e 10). Assim como anteriormente, o valor do banco selecionado é atribuído ao campo DBCAMINHO_DB, o componente destruído e liberado da memória (linhas 11 a 15).

Novamente a **Listagem 10** ilustra um código semelhante. Desta vez é exibido o evento OnClick do botão de salvar o arquivo de backup.

Veja que desta vez, no lugar de declarar um objeto TOpenDialog, foi criado um TSaveDialog, com a finalidade de indicar o local e nome do arquivo a ser salvo (linha 07). Na linha 12 é verificado se o nome do arquivo foi informado com extensão. Caso não tenha sido informada, a linha 13 realiza uma concatenação com '.fbk' a fim de tornar o arquivo válido, caso contrário o campo DBDESTINO_BACKUP apenas recebe o valor do conteúdo selecionado.

Para finalizar a primeira parte do layout, sem ainda abordar as funcionalidades da opção de Restore, definiremos a parte do rodapé do projeto, na qual pode ser visualizada na **Figura 3.** Como pode-se notar, esta é composta por um Panel com alinhamento alBottom, contendo dentro de sim um

Label alinhado alTop e um Memo (memStatus). Com isto a primeira parte de backup está finalizada.

Parâmetro	Descrição
- b	Cria um backup.
-g	Converte as tabelas externas do BD em internas. Quando o banco for restaurado, não haverá mais nenhuma tabela externa, pois elas serão criadas como tabelas convencionais do banco de dados.
-e	Não utiliza a compressão de dados no arquivo gerado.
-fa	Usa um fator de x blocos para uma unidade de fita.
-i	Faz com que os índices não sejam reconstruídos no banco de dados restaurado.
-ig	Ignorar erros de checksum. Usado para fazer backups em bases de dados corrompidas que apresentam problemas de checksum.
-l	Ignorar as transações em limbo (transações em limbo são aquelas que por algum motivo não foram confirmadas nem descartadas).
-m	Faz o backup somente da estrutura do banco (metadata), ignorando os dados que ele possa conter.
-nt	Cria o arquivo de backup em um formato não transportável, ou seja, ele só poderá ser recuperado em um servidor que utilize o mesmo tipo de processador no qual o backup foi gerado.
-ol	Essa opção não deve mais ser utilizada.
-pas	Senha do usuário que se conectará ao servidor para realizar o backup.
-ro	Conecta ao servidor usando um determinado ROLE previamente definido. No momento, essa opção não tem função alguma.
-se	Usa a API de serviços para realizar o backup. Desse modo, quem vai criar o backup nçao vai ser o gbak e sim o próprio Firebird. Note que o arquivo de backup sempre será criado em HDs da máquina onde o servidor IB/FB está rodando
-t	Cria um backup transportável, ou seja, ele poderá ser recuperado em computadores que utilizem um processador diferente do que foi usado na criação do backup – os dados usam o formato XDR.
-u	Usuário que se conectará ao servidor para realizar o backup. Obviamente deve estar previamente definido no servidor.
-v	Reporta na tela todo o processo que está sendo executado durante a operação de backup.
-y	Não mostra mensagens de status na tela, ou redireciona elas para o arquivo ou dispositivo informado em <path>.
-z	Imprime o número da versão do gbak.

Tabela 3. Lista de parâmetros disponíveis para backup Fonte: Livro Firebird 2.0 - Carlos Cantú

Figura 3. Rodapé do layout da aplicação

Listagem 10. Evento OnClick do botão de salvar backup (fbk)

```
01 procedure TForm1.spbFBKClick(Sender: TObject);
02 var
03  svDlgCaminhOFBK : TSaveDialog;
04 begin
05  if not (cdsConfiguracoesDB.State in [dsEdit, dsInsert]) then
06   Exit;
07  svDlgCaminhOFBK := TSaveDialog.Create(nil);
08  try
09   svDlgCaminhOFBK.Filter := 'Arquivos do tipo Backup (.fbk)|*.fbk';
10   if svDlgCaminhOFBK.Execute then
11   begin
12    if ExtractFileExt(svDlgCaminhOFBK.FileName) = '' then
13     cdsConfiguracoesDBDESTINO_BACKUP.AsString :=
       svDlgCaminhOFBK.FileName + '.fbk'
14    else
15     cdsConfiguracoesDBDESTINO_BACKUP.AsString :=  svDlgCaminhOFBK.
       FileName;
16   end;
17  finally
18   FreeAndNil(svDlgCaminhOFBK);
19  end;
20 end;
```

Listagem 11. Evento OnRead do componente JvCreateProcess

```
01 procedure TForm1.JvCreateProcessRead(Sender: TObject; const S: string;
   const StartsOnNewLine: Boolean);
02 begin
03 if Trim(s) <> '' then
04  memStatus.Lines.Add(s)
05 end;
```

Listagem 12. Evento OnExecuto do actBackup

```
01 procedure TForm1.actBackupExecute(Sender: TObject);
02 begin
03  memStatus.Clear;
04  memStatus.Lines.Add('LOG DE BACKUP:');
05  JvCreateProcess.CommandLine :=
06   cdsConfiguracoesAppCAMINHO_GBAK.AsString +
07   ' -user ' + cdsConfiguracoesDBUSUARIO.AsString +
08   ' -pas ' + cdsConfiguracoesDBSENHA.AsString + ' -v ' +
09   getParametrosBackup +
10   '' + cdsConfiguracoesDBCAMINHO_DB.AsString +
11   '' + cdsConfiguracoesDBDESTINO_BACKUP.AsString;
12  JvCreateProcess.Run;
13  while not (JvCreateProcess.State = psReady) do
14   Application.ProcessMessages;
15  Application.MessageBox('Backup realizado com sucesso!',
    'Confirmação!', MB_ICONINFORMATION + MB_OK);
16 end;
```

Antes de continuarmos a implementação, foi utilizado um componente da biblioteca JediVCL (já abordado em outros artigos). Este componente é o JvCreateProcess. Através dele é possível realizar comandos (de prompt de comandos, por exemplo) de forma transparente e o utilizaremos apenas por se tratar de ser um intermediador na hora de obter as mensagens dos processos de Backup/Restore da nossa aplicação. Para que isto ocorra, faremos a implementação no evento OnRead do componente conforme a **Listagem 11**.

Na linha 1 existe uma constante S. Esta constante possuirá o conteúdo referente ao que for obtido nos comandos de Backup e Restore. Na linha 03 verifica-se se a constante possui valor, atribuindo seu valor diretamente às linhas do componente Memo.

Após a implementação, insira no formulário um ActionList (actAcoes), adicionando a ele através de duplo clique duas ações (actBackup e actRestore). Estas serão vinculadas aos botões de Backup e Restore (que será visto logo mais). Codifique o evento onExecute do actBackup conforme a **Listagem 12**.

Extraindo as informações da **Listagem 12**, é possível perceber na linha 05 que o método CommandLine recebe a concatenação do caminho do utilitário gbak, concatenados do usuário e senha informados (linhas 07 e 08), somados dos parâmetros de backup (marcados nos Checkboxes) e por último, o caminho de origem (banco de dados) e destino (arquivo de backup .fbk), conforme as linhas 10 e 11. Caso teríamos neste caso nenhum checkbox habilitado, a instrução montada seria algo semelhante a:

"diretorio Firebird\bin\gbak.exe -user SYSDBA –pas masterkey –v banco de dados.fdb backup.fbk"

Por fim, o procedimento é executado através do método Run (linha 12). As linhas 14 e 15 fazem com que o processo seja aguardado, através do método State que sempre que estiver disponível, estará como psReady. Ao final do processo, a mensagem de confirmação é exibida, conforme a linha 15.

Com isso temos toda a funcionalidade do Backup implementada, deste modo, faremos a configuração do Layout da aba de Restore, conforme ilustrado na **Figura 4**.

Como o procedimento de Restore pode ser realizado em um banco que não seja o mesmo armazenado, não iremos trabalhar com configurações para restauração. Neste caso, conforme é possível notar, adicionamos à aba 15 componentes sendo eles 5 Edits (edtHost, edtLocalFBK, edtLocalFDB, edtUsuario e edtSenha) 6 Labels, 1 ComboBox (cmbPaginacao) e 3 SpeedButtons.

Nota

No lugar do componente da biblioteca Jedi, pode-se utilizar o ShellExecute com o parâmetro SW_HIDE para realizar a chamada do prompt de comandos.

A parte de restauração dos bancos de dados é bem simples e semelhante ao que ocorre com o Backup. Para encurtarmos o exemplo, os parâmetros foram fixados na própria rotina. O evento OnExecute da ação actRestore descrito na **Listagem 13** exibe a implementação responsável pelo Restore do banco. Os parâmetros citados podem ser vistos de forma mais detalhada na **Tabela 4**.

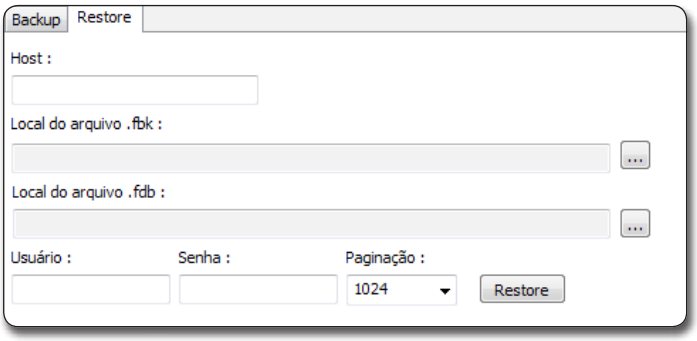


Figura 4. Aba Restore do aplicativo

Nota

As implementações dos SpeedButtons da aba Restore foram ocultadas por serem muito semelhantes a eventos aqui já descritos, porém, o código completo encontra-se disponível para download no portal da DevMedia.

Listagem 13. Evento OnExecute do actRestore

```
01 procedure TForm1.actRestoreExecute(Sender: TObject);
02 var
03   Configuracao : String;
04 begin
05   memStatus.Clear;
06   memStatus.Lines.Add('LOG DE RESTORE:');
07   Configuracao :=
08    cdsConfiguracoesAppCAMINHO_GBAK.AsString +
09    ' -user ' + Trim(edtUsuario.Text) +
10    ' -pas ' + Trim(edtSenha.Text) +
11    ' -r -p ' + cmbPaginacao.Items[cmbPaginacao.ItemIndex] +
12    '' + edtLocalFBK.Text;
13   if Trim(edtHost.Text) = '' then
14    Configuracao := Configuracao + ' localhost:'
15   else
16    Configuracao := Configuracao + '' + Trim(edtHost.Text) + ':';
17   Configuracao := Configuracao + edtLocalFDB.Text;
18   JvCreateProcess.CommandLine := Configuracao;
19   JvCreateProcess.Run;
20   while not (JvCreateProcess.State = psReady) do
21    Application.ProcessMessages;
22   Application.MessageBox('Restore realizado com sucesso!',
     'Confirmação!', MB_ICONINFORMATION + MB_OK);
23 end;
```

Parâmetro	Descrição
-bu	Configura o tamanho do cache utilizado pelo Firebird para o banco de dados restaurado (o valor especificado se dá em páginas de dados).
-c	Cria um banco de dados a partir de um arquivo de backup (restore).
-r	Quando a opção OVERWRITE é omitida, o parâmetro é análogo ao –C. Quando OVERWRITE é definido, o comando é análogo ao –REP.
-i	Faz com que os índices não sejam reconstruídos no banco de dados restaurado.
-k	Restaura um banco de dados, sem no entanto restaurar os arquivos de shadow (espelhos) desse banco.
-mo	Determina se o banco restaurado será atualizável ou não. Access: read_only (somente escrita) ou read_write (escrita/leitura).
-n	Restaura o banco de dados descartando as checks constraints existentes na metadata do banco. Esse processo não inclui os campos declarados como NOT NULL, portanto se você tem campos desse tipo mas alguns registros apresentam NULLs nesses campos, seu banco não poderá ser restaurado.
-o	Restaura uma tabela de cada vez. Geralmente essa opção é usada quando o arquivo de backup está corrompido, assim podemos gerar um banco de dados parcialmente recuperado.
-p	Cria o banco usando páginas com o tamanho determinado por n (em bytes).
-pas	Senha do usuário que se conectará ao servidor para realizar o restore. Esse usuário passará a ser o "dono" do banco de dados restaurado.
-rep	Se o restore especificar um banco de dados que já existe, ele será substituído (sobreposto) pelo banco que está sendo restaurado. Essa opção é perigosa, pois caso haja alguma conexão com o banco enquanto ele está sendo restaurado, poderá corrompê-lo. Se usada, tenha certeza de que nenhuma conexão com o banco ocorrerá durante o tempo em que o processo estiver sendo executado.
-se	Usa a API de serviços para realizar o backup. Desse modo, quem vai criar o backup não vai ser o gbak e sim o próprio servidor. Note que o arquivo de backup sempre será criado em HDs da máquina onde o servidor IB/FB está rodando.
-use	Durante um restore, o padrão é que 20% do espaço de uma página do banco fique livre para ser utilizado posteriormente. Caso esse parâmetro seja informado, todo o espaço da página será utilizado durante o restore. Isso é interessante para bancos Read-Only, pois diminui o tamanho do arquivo do BD.
-user	Usuário que se conectará ao servidor para realizar o backup/restore. Obviamente deve estar previamente definido no servidor.
-v	Reporta na tela todo o processo que está sendo executado durante a operação de backup/restore.
-y	Não mostra mensagens de status na tela, ou redireciona elas para o arquivo ou dispositivo informado.
-z	Imprime o número da versão do gbak.

Tabela 4. Lista de parâmetros para restauração do banco - Fonte: Livro Firebird 2.0 - Carlos Cantú

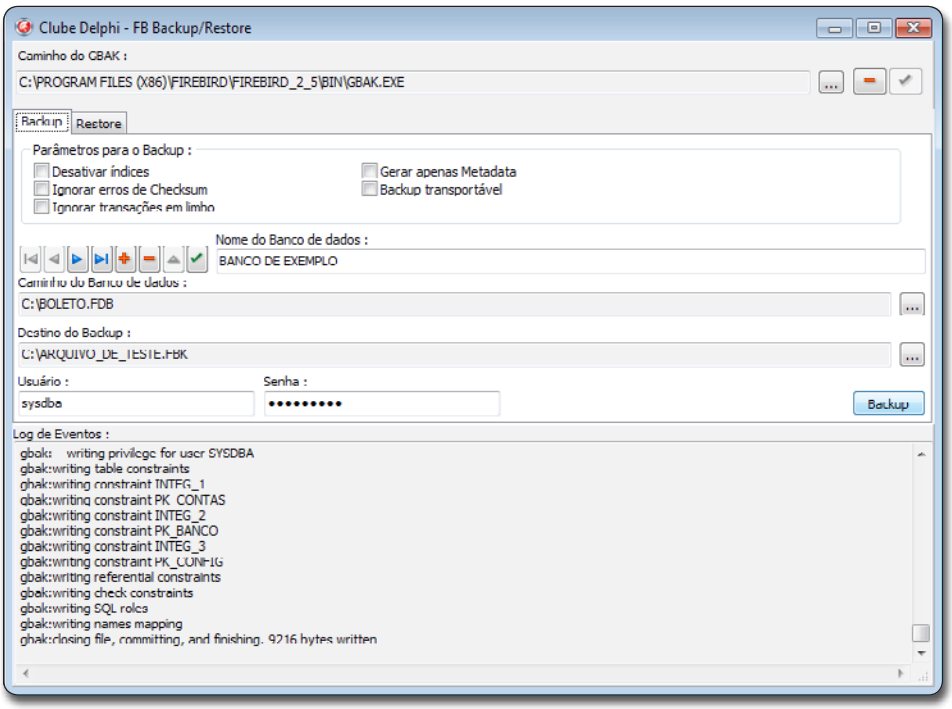


Figura 5. Aplicação em execução

Listagem 14. Função CanDoRestore

```
01 function TForm1.CanDoRestore: Boolean;
02 begin
03  Result :=
04   ( not(Trim(edtLocalFBK.Text) = '') and
05    not(Trim(edtLocalFDB.Text) = '') and
06    not(Trim(edtUsuario.Text) = '') and
07    not(Trim(edtSenha.Text) = ''));
08 end;
```

Listagem 15. Validações do evento OnUpdate do actionList

```
01 procedure TForm1.actAcoesUpdate(Action: TBasicAction; var Handled:
Boolean);
02 begin
03  actBackup.Enabled  := CanDoBackup;
04  actRestore.Enabled := CanDoRestore;
05 end;
```

Na **Listagem 13** temos um código semelhante ao evento de realização do Backup. Note que novamente os parâmetros são concatenados entres as linhas 07 a 12 e atribuídas à variável Configuracao. A diferença aqui no processo se dá no preenchimento do campo edHost. Caso o mesmo não seja preenchido, o código aponta o host como localhost: (linha 14), caso contrário, a linha 16 concatena o Host informado. Dai em diante o processo é realizado da mesma maneira, executando o método Run do JvCreateProcess (linha 19) e aguardando o retorno como psReady, exibindo imediatamente o progresso no componente Memo e finalmente mostrando uma mensagem de que o processo foi realizado.

Com isso finalizamos o processo de Backup e Restore. As demais configurações podem ser realizadas para efeito de melhoria e prevenção de erros, neste caso, assim como a situação da função CanDoBackup, criamos a função CanDoRestore que válida o conteúdo dos Edits obrigatórios para preenchimento (**Listagem 14**) e por último, o evento que controlará o Enabled dos botões de Backup e Restore, inserido no ActionList do formulário (**Listagem 15**).

Na **Listagem 14** podemos ver que nada mais é feito do que simplesmente verificar se os campos possuem conteúdo, estando os Edits preenchidos, o retorno da função CanDoRestore será True.

Na **Listagem 15** podemos notar o uso das duas funções anteriormente criadas, fazendo com que os botões sejam automaticamente habilitados/desabilitados de acordo com os preenchimentos das configurações especificadas. Por fim, a **Figura 5** ilustra a aplicação em execução e funcionamento.

Conclusão

Com isso finalizamos o nosso exemplo. Como foi possível notar, realizar a automatização dos seus processos criando interfaces para tornar as nossas rotinas mais amigáveis não é algo tão complexo. Com um pouco de criatividade é possível adaptar este exemplo para realizar inúmeras outras tarefas como a de compactar o arquivo .fbk, realizar o Upload do mesmo para um FTP, criar backups agendados, backups incrementais, confirmação de backup por e-mail entre outras tantas funcionalidades. Mais uma vez, deixo a cargo do leitor a tarefa de implementar algumas destas funcionalidades, elaborando um aplicativo mais ideal para o seu ambiente.

Daniel Sobrinho Laporte
daniel.laporte@gmail.com/Twitter: @daniel_laporte
Colunista e editor técnico da revista Clube Delphi. Graduado em Sistemas de informação e pós-graduado em administração de bancos de dados. Trabalha como desenvolvedor e consultor independente. Desenvolve em Delphi desde 2005 adotando tanto a arquitetura cliente-servidor como multicamadas e os mais variados bancos de dados.

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

FastReport

Elaboração de Relatórios Passo a Passo – Final

Este artigo faz parte de um curso

Resumo DevMan

De que se trata o artigo:

Este artigo sobre FastReport demonstra outros recursos interessantes e avançados desta ferramenta que podem ser úteis no dia-a-dia de acordo com a necessidade do projeto. Serão apresentados recursos como variáveis, eventos, cross-tab, impressão de arquivos, criação e exportação de relatórios em tempo de execução, dentre outros.

Em que situação o tema é útil:

Utilizar o FastReport para elaborar relatórios de formas diferentes à tradicional, bem como conhecer e empregar alguns recursos avançados desta ferramenta, como é o caso da elaboração de relatórios em tempo de execução.

FastReport :

Ter a disposição relatórios que possibilitem diversas visões da empresa é algo imprescindível para os usuários. Sendo assim, contar com ferramentas que auxiliem e agilizem ao máximo o processo de criação, edição, visualização e impressão de relatórios, como o FastReport, é um fator extremamente importante no âmbito de desenvolvimento de Software. Presente no Delphi XE2, esta poderosa ferramenta oferece uma diversidade de recursos que vão além da elaboração de relatórios tradicionais baseados em DataSets, onde o desenvolvedor pode criar relatórios de forma dinâmica e preenchê-los de acordo com sua necessidade. Neste artigo serão apresentadas outras características do FastReport que podem ser úteis em diversos momentos, bem como a adição de recursos e ampliação das possibilidades no relatório de exemplo criado no artigo anterior.

No artigo anterior sobre a ferramenta FastReport, foi abordada a construção tradicional de relatórios contendo três níveis (Master/Detail/SubDetail) com base em DataSets e outros recursos interessantes, como é o caso do Drill-Down/Drill-Up, que possibilita aumentar/diminuir o nível de detalhamento. Neste artigo, serão abordados alguns dos diversos recursos adicionais que o FastReport oferece, como é o caso de variáveis, eventos, elaboração de relatórios cross-tab e etc. As variáveis permitem que o desenvolvedor envie valores para o FastReport em tempo de execução, preenchendo os recipientes conforme necessitar. Os eventos estão presentes em grande parte dos objetos que compõem o relatório, permitindo a realização de tarefas antes ou após a impressão dos componentes, dentre outros momentos. O recurso cross-tab consiste simplesmente na elaboração de relatórios que facilitam o cruzamento de informações, que podem ser baseadas em diversas tabelas. Também existe a possibilidade de montar um relatório em tempo de execução, diretamente através de código fonte, utilizando o componente principal TfrxReport, que oferece a base para a elaboração do relatório. Esta ferramenta oferece ainda a alternativa de imprimir arquivos, onde os mesmos podem ser carregados facilmente através do Delphi e suas informações enviadas para o relatório. Veremos isto mais adiante.

As funcionalidades do FastReport mencionadas, dentre muitas outras, aumentam o número de possibilidades e permitem uma maior flexibilidade para todo o processo relacionado a relatórios, tornando-o uma excelente escolha para os projetos.

Tutorial

Variáveis

O FastReport oferece suporte a variáveis. Este recurso é muito útil quando existe a necessidade de utilizar valores dinâmicos em relatórios, tais como o envio de informações em tempo de execução. As variáveis podem ser previamente declaradas, através do designer, ou desconhecidas, onde os valores das mesmas precisam ser informados posteriormente.

Para dar início a um exemplo prático de como trabalhar com variáveis em relatórios, o mesmo projeto criado no artigo anterior pode ser utilizado. Tal projeto já possui um simples relatório de contas a receber baseado em três níveis (Master/Detail/SubDetail – Pessoa/Conta/Parcelas) que foi definido. Após o carregamento do mesmo, o Designer do FastReport contendo o relatório deve ser aberto, lembrando que este processo é efetuado através de um clique duplo sob o objeto frxReport1. Para dar continuidade ao exemplo, uma variável denominada "Usuario" deve ser declarada, cujo intuito será preenchê-la em tempo de execução com o nome do usuário que está efetuando a impressão. Sendo assim, o menu

Report>Variables deve ser acessado. Neste momento, a janela "Edit Variables" é exibida e através da mesma, o desenvolvedor pode facilmente declarar as variáveis para serem utilizadas no relatório. O primeiro passo é adicionar uma categoria através do botão "Category" do editor. Em seguida, deve-se criar a variável através do botão "Variable". Para renomear categorias e variáveis, o desenvolvedor pode utilizar o botão "Edit" ou simplesmente a tecla F2. A **Figura 1** apresenta a janela "Edit Variables" contendo a categoria e a variável criada.

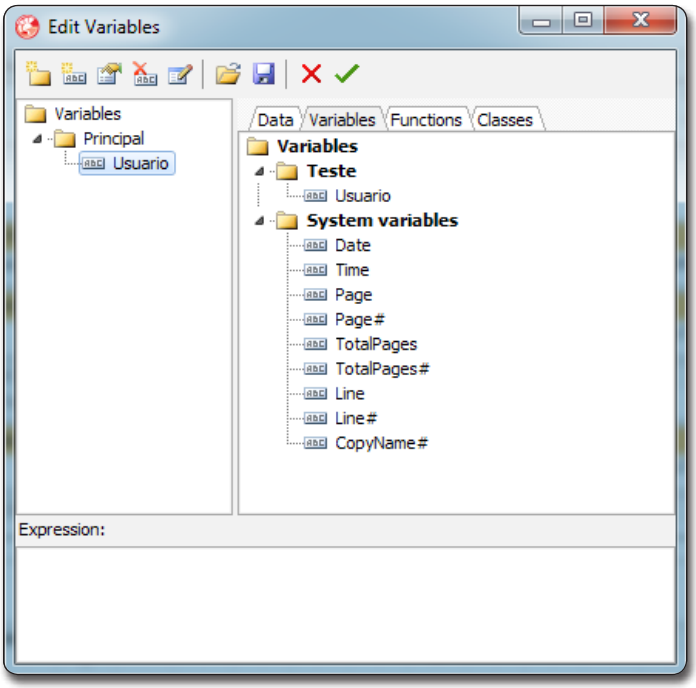


Figura 1. Janela "Edit Variables" com a definição da variável "Usuario"

Criada a variável, é necessário definir o local onde o valor da mesma será exibido. Neste exemplo foi adicionado um objeto MemoView através do botão "Text Object" da paleta de ferramentas, na banda ReportSummary1, criada no artigo anterior. Em seguida, para exibir o futuro valor da variável basta efetuar um clique duplo sob o objeto MemoView inserido, o que acarretará na abertura da janela "Memo". Nesta janela, deve-se clicar no botão "Insert Expression" e novamente efetuar um clique duplo sob a variável desejada, presente na aba "Variables", finalizando o processo através do botão OK. Vale ressaltar que o desenvolvedor também pode definir a exibição do valor de uma variável inserindo diretamente a expressão "[nome_variável]", ou ainda efetuar combinações como "Emitido por: [Usuario], no dia [Date]", onde a variável "Usuario" é substituída pelo seu valor e a função "Date" é utilizada para exibir a data atual.

Terminada a definição da variável através do Designer do Fast Report, o próximo passo é modificar o código fonte que efetua o carregamento do relatório, também definido anteriormente, para que o valor da mesma seja informado. A **Listagem 1** apresenta o código fonte do evento OnClick do botão alterado para definir o valor da variável "Usuario".

Após o início da conexão com o banco de dados através do objeto SQLConnection1, os DataSets também são iniciados, de forma equivalente ao código definido no artigo anterior (linhas 03 a 07). Entretanto, foram inseridas duas novas linhas. A primeira, frxReport1.Variables[], é utilizada para adicionar determinado valor a uma variável, onde neste caso a variável "Usuario" recebe o valor de "Giuliano" (linha 08). O valor da mesma deve ser definido com a utilização da função "QuotedStr", que por sua vez retorna uma String entre aspas. Em seguida, a segunda linha adicionada apresenta o método PrepareReport (linha 09), onde o relatório é devidamente preparado. Para concluir, o mesmo é exibido através de ShowReport (linha 10).

Listagem 1. Código do evento OnClick() do botão btExibirRelatorio

```
01 procedure TFoPrincipal.btExibirRelatorioClick(Sender: TObject);
02 begin
03   if not SQLConnection1.Connected then
04     SQLConnection1.Connected := True;
05   sdsPessoa.Open;
06   sdsContasReceber.Open;
07   sdsParcelas.Open;
08   frxReport1.Variables['Usuario'] := QuotedStr('Giuliano');
09   frxReport1.PrepareReport;
10   frxReport1.ShowReport;
11 end;
```

Existem outras maneiras de criar variáveis para o relatório, tais como o uso da classe TfrxVariable, presente na Unit frxVariables ou o uso da função Variables.Add, presente na classe TfrxReport. Também existem outras formas de definir os valores para as variáveis criadas, utilizando o método OnGetValue de um objeto TfrxReport, por exemplo, onde tal método é chamado pelo FastReport quando o mesmo se depara com uma variável não definida ou desconhecida.

Como é possível observar, a utilização de variáveis é um recurso muito importante para elaboração de relatórios dinâmicos.

Eventos internos

Outro recurso que pode ser muito útil ao utilizar o FastRport é a utilização de eventos. A maior parte dos objetos que fazem parte de um relatório como páginas, bandas, Memos e etc, também possuem diversos eventos que podem ser definidos através da aba "Events" do Designer. Sendo assim, o desenvolvedor tem a opção de atribuir determinado código para conseguir alguma funcionalidade que não esteja presente por padrão ou para efetuar cálculos manualmente, dentre outros recursos. Para iniciar um simples exemplo de como definir eventos para os objetos do relatório, o mesmo projeto criado no artigo anterior pode ser utilizado. Em seguida, o Designer também deve ser aberto através do objeto frxReport1. Neste momento, será criado um simples filtro para exibir as informações no relatório somente quando o ID da pessoa for igual a "3", apenas para ilustrar como utilizar os eventos, uma vez que convém implementar esta funcionalidade diretamente via código SQL. Sendo assim, a banda MasterData1 deve ser selecionada e na aba "Events", deve-se efetuar um clique duplo sobre o evento OnBeforePrint, para a definição de um código que deve ser executado todas as vezes antes da impressão da mesma. A **Listagem 2** apresenta o código que foi definido para o evento OnBeforePrint da banda MasterData1.

Listagem 2. Código do evento OnBeforePrint da banda MasterData1

```
01 procedure MasterData1OnBeforePrint(Sender: TfrxComponent);
02 begin
03  if (<frxdbPessoa."ID"> = '3') then
04  begin
05    MasterData1.Visible := True;
06    DetailData1.Visible := True;
07    GroupHeader1.Visible := True;
08    GroupHeader1.DrillDown := False;
09    SubdetailData1.Visible := True;
10    GroupFooter1.Visible := True;
11  end
12  else
13  begin
14    MasterData1.Visible := False;
15    DetailData1.Visible := False;
16    GroupHeader1.Visible := False;
17    GroupHeader1.DrillDown := False;
18    SubdetailData1.Visible := False;
19    GroupFooter1.Visible := False;
20  end;
21 end;
```

O código adotado foi o Pascal Script. Entretanto o desenvolvedor pode utilizar outras linguagens, como é o caso do C++Script. É possível notar também que no início do bloco é efetuado um teste para verificar se o ID presente no objeto frxdbPessoa é equivalente a "3" (linha 03) e se tal condição é satisfeita, então todas as bandas são exibidas (linhas 05 a 10), bem como a funcionalidade DrillDown é desativada, lembrando que esta funcionalidade pode ser utilizada para aumentar/diminuir o nível de detalhamento das informações. Caso a condição não seja satisfeita, então todas as bandas são ocultadas (linhas 14 a 19).

Evidentemente este é um simples exemplo da utilidade dos eventos, que por sua vez vai muito além do que foi apresentado. Outro simples exemplo seria iniciar uma nova página toda vez que uma banda fosse exibida, onde o desenvolvedor poderia simplesmente definir o evento OnAfterPrint da mesma com o código Engine.NewPage, que o FastReport geraria automaticamente uma nova página, cuja função é semelhante a propriedade StartNewPage quando definida como True.

Exportação de relatórios

É possível permitir ao usuário exportar seu relatório para o Excel, por exemplo, onde o mesmo pode implementar quantos filtros desejar, ou outro formato de arquivo suportado, como é o caso de PDF, BMP, XML, HTML, etc. Os componentes para a exportação se encontram na aba FastReport 4 Exports. Para implementar esta funcionalidade em um relatório, basta que o desenvolvedor adicione um ou mais componentes referentes aos formatos que o mesmo deseja exportar no formulário em questão. Sendo assim, um novo botão será adicionado automaticamente na janela de pré-visualização do relatório, denominado "Export", onde o usuário pode simplesmente clicar, definir algumas propriedades e exportar o mesmo.

Como é possível observar, o FastReport implementa esta funcionalidade de uma maneira muito fácil e rápida. O desenvolvedor ainda pode exportar o relatório para um formato diretamente através de código fonte, sem que exista uma pré-visualização do mesmo ou uma intervenção do usuário. Para demonstrar este recurso, o mesmo projeto anterior pode ser utilizado e um novo botão deve ser adicionado ao formulário. Neste exemplo, o relatório será exportado para PDF, sendo assim, a Unit frxExportPDF deve ser inclusa na seção Uses do formulário principal. A **Listagem 3** apresenta o código para a exportação manual que foi definido no evento OnClick do novo botão inserido, btExportarRelatorio.

Listagem 3. Código do evento OnClick do botão btExportarRelatorio

```
01 procedure TFoPrincipal.btExportarRelatórioClick(Sender: TObject);
02 var
03   ExpPDF: TfrxPDFExport;
04   LocalArquivo: String;
05 begin
06   ExpPDF := TfrxPDFExport.Create(nil);
07   LocalArquivo := 'C:\RelatorioContasReceber.pdf';
08   with ExpPDF do
09   begin
10     FileName := LocalArquivo;
11     ShowDialog := False;
12   end;
13   frxReport1.Variables['Usuario'] := QuotedStr('Giuliano');
14   frxReport1.PrepareReport;
15   frxReport1.Export(ExpPDF);
16 end;
```

É possível notar que inicialmente são declaradas duas variáveis, uma do tipo TfrxPDFExport, o componente responsável pela exportação em PDF, e a LocalArquivo (linhas 03 e 04). No início do bloco begin, o objeto do tipo TfrxPDFExport (ExpPDF) é devidamente criado (llinha 06) e a variável LocalArquivo recebe o endereço físico onde o arquivo exportado será gravado (linha 07). Em seguida, tal local de gravação é definido para o objeto ExpPDF através de sua propriedade FileName (linha 10), e a exibição da janela de pré-definição das propriedades de exportação é configurada para não ser exibida através da propriedade ShowDialog definida como False (linha 11). Como a variável "Usuario" foi criada anteriormente, ela também é informada para evitar possíveis erros (linha 13). Para concluir, o relatório é preparado (Prepare Report – linha 14) e exportado através do método Export (linha 15) que recebe o objeto de exportação em PDF como parâmetro.

Como é possível observar, o FastReport é bem flexível e na maioria dos casos, sempre oferece várias maneiras de efetuar a mesma tarefa, cabendo ao desenvolvedor utilizar a mais adequada.

Relatório Cross-Table

Um relatório do tipo Cross-Table, também conhecido como Cross-Tab ou Tabela Cruzada, é composto por linhas e colunas, onde as informações são cruzadas, normalmente totalizando determinados valores e apresentando outra forma de visualização para as informações, possibilitando a realização de comparações. Os mesmos se assemelham as tradicionais Planilhas Eletrônicas. A **Figura 2** apresenta um simples exemplo de Cross-Table.

Para dar início à criação de um relatório Cross-Table com base no FastReport, o mesmo projeto utilizado no artigo anterior deve ser reaberto. Nele foi criado um banco de dados baseado no Firebird com as tabelas Pessoa, ContasReceber e Parcelas para a execução do primeiro exemplo, e que tal projeto já se encontra com as configurações básicas para que este exemplo possa ser efetuado, tal como as definições do objeto SQLConnection1, responsável pela conexão com o banco.

Para dar continuidade a este exemplo, um novo objeto TSimpleDataSet deve ser adicionado ao formulário. O componente TSimpleDataSet do Framework DBExpress é utilizado para executar uma instrução SQL através de um objeto TSQLConnection, armazenando as informações em um Buffer na memória para oferecer suporte as operações de CRUD (Create/Read/Update/Delete). O TSimpleDataSet inserido recebeu a denominação de sdsCross. Sua propriedade Connection deve ser definida com o valor de SQLConnection1. Para concluir a configuração do mesmo, basta inserir o código SQL que será utilizado para a apresentação deste exemplo na propriedade DataSet.CommandText. A **Listagem 4** apresenta o código SQL utilizado para o objeto sdsCross.

É realizado um Join nas três tabelas (Pessoa/ContasReceber/Parcelas) com o intuito de obter as informações para que o exemplo possa ser realizado.

O próximo passo é adicionar ao formulário principal, um objeto do tipo TfrxDBDataSet. Este componente permite que objetos do tipo TDataSet ou TDataSource, presentes no formulário, possam ser ligados e assim utilizados em um relatório do FastReport. Ele recebeu a denominação de frxdbCross e sua propriedade DataSet foi definida com o valor de sdsCross.

Para criar o relatório Cross-Table, um objeto do tipo TfrxReport deve ser adicionado ao formulário e posteriormente, o Designer do FastReport deve ser aberto através de um clique duplo sob o

	2010	2011	2012	Total
Janeiro	10	20	30	60
Fevereiro	15	5	10	30
Março	20	35	40	95
Abril	30	20	30	80
Maio	10	5	15	30
Junho	10	10	20	40
Total	95	95	145	335

Figura 2. Exemplo de Cross-Table

Listagem 4. Código SQL para o objeto sdsCross

```
SELECT
  PARCELAS.*,
  PESSOA.NOME
FROM PARCELAS
INNER JOIN CONTASRECEBER
  ON CONTASRECEBER.ID = PARCELAS.IDCONTASRECEBER
INNER JOIN PESSOA
  ON PESSOA.ID = CONTASRECEBER.IDPESSOA
```

mesmo, lembrando que, por padrão, tudo que é realizado através do Designer fica armazenado no arquivo .dfm do formulário em questão. Dentro do Designer, deve-se acessar o menu Report e selecionar a opção Data. É possível notar que é exibida uma janela contendo uma lista de objetos que podem ser empregados para a interligação com os componentes do formulário, sendo assim, o frxdbCross deve ser selecionado para que os campos do sdsCross possam ser utilizados no relatório. Terminada esta etapa, deve-se inserir um objeto do tipo Cross-Tab no relatório através do botão "DB Cross-Tab object" (), presente na barra de objetos do Designer do FastReport. Ao inserir o mesmo, a janela "Cross-Tab Editor" é automaticamente aberta para a definição dos campos e das características do Cross-Table. A **Figura 3** apresenta as configurações que foram definidas para este exemplo através do Cross-Tab Editor.

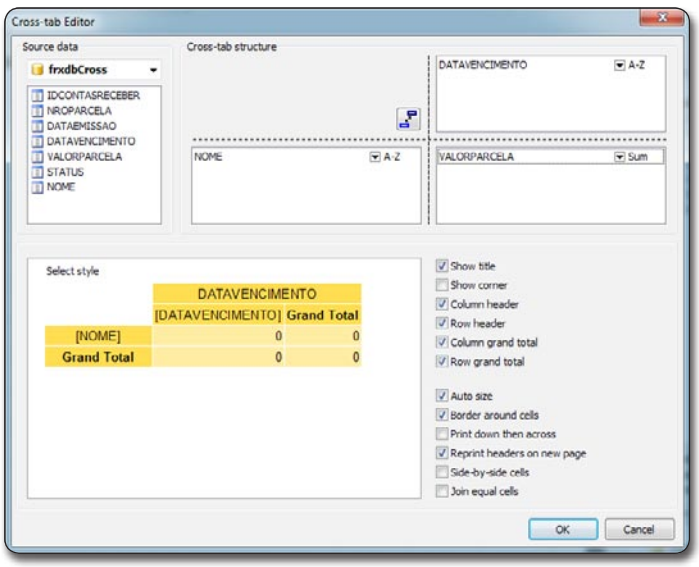


Figura 3. Definições untilizadas no Cross-Tab Editor para o Exemplo

É possível notar a existência do agrupamento "Source data", contendo todos os campos de acordo com o objeto selecionado, que neste exemplo é o frxdbCross. Para definir a estrutura do Cross-Table basta arrastar os campos desejados para seus respectivos recipientes presentes no agrupamento "Cross-tab Structure", que neste exemplo foram NOME, DATAVENCIMENTO e VALORPARCELA. É notável que várias funções estatísticas estão disponíveis para este relatório, como é o caso de soma e contagem por exemplo. O Editor também permite que o relatório possa ser personalizado, trazendo diversas propriedades que podem ser ativadas ou desativadas, possuindo também uma coleção de temas que podem ser aplicados através do botão "Select style", onde neste projeto o tema selecionado foi "Orange". Terminada as definições do Cross-Table, basta clicar no botão OK e fechar o Designer do FastReport.

De volta ao Delphi, foi inserido um TSpeedButton para que os objetos necessários e o relatório sejam carregados no momento em que o mesmo for clicado. A **Listagem 5** apresenta o código para o evento OnClick do TSpeedButton adicionado.

É efetuado um teste para verificar se o objeto SQLConnection1 encontra-se ativo através de sua propriedade Connected (linha 03) e caso não esteja, o mesmo é ativado em seguida na linha 04. Para concluir, o sdsCross é aberto e o relatório exibido através do método ShowReport (linhas 05 e 06 respectivamente). A **Figura 4** apresenta o visual do relatório com alguns dados de demonstração.

Listagem 5. Código do evento OnClick do botão adicionado

```
01 procedure TFoPrincipal.btExibirRelatorioCrossTableClick(Sender: TObject);
02 begin
03   if not SQLConnection1.Connected then
04     SQLConnection1.Connected := True;
05   sdsCross.Open;
06   frxReport2.ShowReport;
07 end;
```

	Data de Vencimento				
	20/08/2012	20/09/2012	20/10/2012	20/11/2012	Total
PESSOA DE TESTE 1		550	550	300	1400
PESSOA DE TESTE 3	500				500
Total	500	550	550	300	1900

Figura 4. Exibição do relatório Cross-Table

Criando um relatório Cross-Table em tempo de execução

Outra possibilidade que o desenvolvedor pode empregar em seus projetos é a criação de relatórios em tempo de execução. Para este exemplo, também foi criado um relatório do tipo Cross-Table, porém o mesmo foi preenchido manualmente utilizando um objeto do tipo TfrxCrossView. Este componente do FastReport

permite que o desenvolvedor crie seu Cross-Table diretamente da forma que desejar, sem que exista a necessidade de efetuar um vínculo com uma ou mais tabelas do banco de dados através de DataSets. Sendo assim, para dar continuidade ao exemplo, o primeiro passo é criar um novo procedimento auxiliar para manipular o evento OnBeforePrint do objeto TfrxReport, com o intuito de preencher o objeto TfrxCrossTable manualmente antes da pré-visualização e impressão do relatório. A **Listagem 6** apresenta o procedimento que foi criado no formulário principal com este objetivo.

Foi declarada uma variável do tipo TfrxCrossView para auxiliar durante a inclusão dos valores (linha 03). Após o início do bloco, é efetuado um teste para verificar se o componente atual é do tipo TfrxCrossView (linha 05), e caso seja verdadeiro, seu valor é atribuído a variável criada anteriormente (CrossView – linha 07). Para concluir, através do método AddValue deste mesmo objeto, as células são preenchidas, uma vez que tal método possibilita a inserção manual de valores para o Cross-Table (linhas 10 a 15). No método AddValeu o primeiro parâmetro deste corresponde a linha, o segundo a coluna, e o terceiro seu devido valor.

O próximo passo para o projeto é inserir um novo botão no formulário, cujo intuito será a criação do relatório em tempo de execução. A **Listagem 7** apresenta o código definido para o evento OnClick do botão.

São declaradas variáveis de acordo com as classes e componentes do FastReport que serão necessários (linhas 03 a 06). O TfrxReport é o componente principal responsável pelo relatório, o TfrxReportPage é utilizado para a criação de páginas para o relatório, o TfrxReportTitle permite a utilização de uma banda do tipo título e o TfrxCrossView, conforme mencionado, é o responsável pela geração do Cross-Table. Após o início do bloco, o objeto TfrxReport é criado, bem como a página para o mesmo (linha 08 e 09). Em seguida (linha 12), é atribuído automaticamente um nome único ao objeto Pagina através do método CreateUniqueName. Também é empregado o método SetDefaults (linha 13) cuja função é carregar os valores padrões para a página, tal como papel, orientação, etc. Posteriormente, é criada a banda do tipo título para a página e a mesma tem suas propriedades definidas, tais como nome, posicionamento e tamanho (linha 15 a 22). Na linha 23 é criado um objeto do tipo Memo (TfrxMemoView) para a banda de título com o objetivo de exibir um texto, que por sua vez também teve suas propriedades definidas (linhas 25 a 31). Em seguida, o objeto TfrxCrossView é criado para a página e também é definido um nome único para o mesmo (CreateUniqueName – linhas 33 e 34 respectivamente). Para concluir, o evento OnBeforePrint do relatório é definido para o procedimento criado anteriormente, RelatorioBeforePrint (linha 37), sendo o relatório posteriormente preparado (PrepareReport – linha 38), exibido (ShowReport – linha 39) e liberado da memória após sua exibição (FreeAndNil – linha 41).

Listagem 6. Código do procedimento RelatorioBeforePrint

```
01 procedure TFoPrincipal.RelatorioBeforePrint(Componente:
     TfrxReportComponent);
02 var
03   CrossView: TfrxCrossView;
04 begin
05   if Componente is TfrxCrossView then
06   begin
07     CrossView := TfrxCrossView(Componente);
08     with CrossView do
09     begin
10       AddValue(['Janeiro'], ['2011'], [10]);
11       AddValue(['Janeiro'], ['2012'], [40]);
12       AddValue(['Fevereiro'], ['2011'], [15]);
13       AddValue(['Fevereiro'], ['2012'], [30]);
14       AddValue(['Março'], ['2011'], [20]);
15       AddValue(['Março'], ['2012'], [30]);
16     end;
17   end;
18 end;
```

Listagem 7. Criação do relatório em tempo de execução

```
01 procedure TFoPrincipal.btRelatorioTempoExecucaoClick(Sender: TObject);
02 var
03   Report: TfrxReport;
04   Pagina: TfrxReportPage;
05   Titulo: TfrxReportTitle;
06   CrossView: TfrxCrossView;
07 begin
08   Report := TfrxReport.Create(nil);
09   Pagina := TfrxReportPage.Create(Report);
10   with Pagina do
11   begin
12     CreateUniqueName;
13     SetDefaults;
14   end;
15   Titulo := TfrxReportTitle.Create(Pagina);
16   with Titulo do
17   begin
18     CreateUniqueName;
19     Top := 0;
20     Height := 60;
21     Width := Pagina.Width;
22   end;
23   with TfrxMemoView.Create(Titulo) do
24   begin
25     CreateUniqueName;
26     Text := 'Exemplo Cross-Table';
27     Height := 15;
28     Align := baClient;
29     VAlign := vaCenter;
30     HAlign := haLeft;
31     Font.Size := 18;
32   end;
33   CrossView := TfrxCrossView.Create(Pagina);
34   CrossView.CreateUniqueName;
35   with Report do
36   begin
37     OnBeforePrint := RelatorioBeforePrint;
38     PrepareReport;
39     ShowReport;
40   end;
41   FreeAndNil(Report);
42 end;
```

Como pode ser observado, os relatórios em FastReport podem ser facilmente criados em tempo de execução, oferecendo ao desenvolvedor diversas maneiras de empregá-los em seus projetos.

Impressão de arquivos

Além da criação, edição e impressão de relatórios, o FastReport também funciona como uma ferramenta para impressão de outros recursos, como é o caso de arquivos, textos e etc. Este processo pode ser efetuado com o auxílio do objeto TStringList, que permite o carregamento de uma lista de Strings, e o evento OnGetValue presente em um objeto do tipo TfrxReport. O evento OnGetValue possibilita que um valor dinâmico seja passado para o relatório, podendo ser utilizado por exemplo, para enviar valores de variáveis conforme mencionado anteriormente. Sendo assim, cada vez que uma variável desconhecida está presente no relatório, o FastReport efetua uma chamada a este evento, que por sua vez, retorna o valor da mesma, onde evidentemente, o desenvolvedor deve informar tal valor.

Para dar início a mais um exemplo prático com o FastReport para a impressão de arquivos, o primeiro passo é criar um procedimento para o método OnGetValue, uma vez que neste exemplo o relatório também será criado em tempo de execução. A **Listagem 8** apresenta o código para o procedimento auxiliar.

Este procedimento, por padrão, deve receber dois parâmetros conforme foi definido, um para o nome da variável (VarName) e outro para seu valor (Value). Como o evento OnGetValue é chamado toda vez que existe uma variável desconhecida, a função CompareText (linha 06), que efetua a comparação entre dois textos e retorna o valor "0" se ambos são iguais, pode ser empregada para verificar se o valor da variável desconhecida atual (VarName) é equivalente a determinado valor (conteudo). Se a condição anterior é satisfeita, então é criado um objeto TStringList (Texto – linha 08). Em seguida, o método LoadFromFile (linha 09) é chamado para possibilitar o carregamento de um arquivo externo para o objeto. Neste exemplo, foi criado um arquivo de texto (Arquivo .txt) contendo algumas palavras para possibilitar o teste, sendo o mesmo gravado no diretório em que se encontra o executável deste exemplo. Para concluir, o valor da propriedade Text do objeto TStringList, que por sua vez contém o texto carregado do arquivo, é atribuído ao parâmetro Value (linha 10), passando para o FastReport o valor da variável "conteudo", sendo a TStringList liberada da memória (Free – linha 11) logo em seguida.

Terminado o procedimento auxiliar que será chamado em tempo de execução, o próximo passo é inserir um novo botão ao formulário com o intuito de criar o relatório de impressão de arquivo no momento em que o usuário clicar sob o mesmo. A **Listagem 9** demonstra o código do evento OnClick para o último botão inserido.

Foram declaradas três variáveis (linha 03 a 05), uma do tipo TfrxReport, o componente principal para o relatório, uma do tipo TfrxReportPage, para a página do mesmo, e uma do tipo Tfrx MasterData, para a criação de uma banda do tipo MasterData para a página. Em seguida, após o início do bloco, o relatório (Report) e sua página (Pagina) são criados (linhas 08 e 09). O objeto Pagina recebe um nome único (CreateUniqueName – linha 12) e são carregados seus valores padrões (SetDefaults – linha 13). Posteriormente, a banda MasterData é criada para a Pagina e suas propriedades

Listagem 8. Código do procedimento RelatorioGetValue

```
01 procedure TFoPrincipal.RelatorioGetValue(const VarName: String;
02   var Value: Variant);
03 var
04   Texto: TStringList;
05 begin
06   if CompareText(VarName, 'conteudo') = 0 then
07   begin
08     Texto := TStringList.Create;
09     Texto.LoadFromFile('Arquivo.txt');
10     Value := Texto.Text;
11     Texto.Free;
12   end;
13 end;
```

Listagem 9. Código do evento OnClick do botão btImprimirValorArquivo

```
01 procedure TFoPrincipal.btImprimirValorArquivoClick(Sender: TObject);
02 var
03   Report: TfrxReport;
04   Pagina: TfrxReportPage;
05   Banda: TfrxMasterData;
06 begin
07   try
08     Report := TfrxReport.Create(nil);
09     Pagina := TfrxReportPage.Create(Report);
10     with Pagina do
11     begin
12       CreateUniqueName;
13       SetDefaults;
14     end;
15     Banda := TfrxMasterData.Create(Pagina);
16     with Banda do
17     begin
18       CreateUniqueName;
19       Top := 0;
20       Height := 300;
21       Width := Pagina.Width;
22       Banda.RowCount := 1;
23     end;
24     with TfrxMemoView.Create(Banda) do
25     begin
26       CreateUniqueName;
27       Text := '[conteudo]';
28       Height := 100;
29       Width := Pagina.Width;
30       Align := baClient;
31       VAlign := vaCenter;
32       HAlign := haLeft;
33       Font.Size := 18;
34     end;
35     with Report do
36     begin
37       OnGetValue := RelatorioGetValue;
38       PrepareReport;
39       ShowReport;
40     end;
41   finally
42     FreeAndNil(Report);
43   end;
44 end;
```

CURSOS ONLINE

.net magazine

A Revista .net Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

MAIS de 20 CURSOS DISPONÍVEIS

CONHEÇA OS CURSOS MAIS RECENTES:

- Curso Básico de C#
- C# 5 e suas novidades
- Integrando Aplicativos ASP>NET C# com redes sociais (Twitter e Facebook)
- Refatoração com C# .NET
- ASP.NET MVC – Sistema de Vestibular

Para mais informações :

www.devmedia.com.br/curso/netmagazine

(21) 3382-5038

http://www.devmedia.com.br

DevMedia

como nome, posicionamento (Top) e tamanho (Height / Width) são definidas (linhas 16 a 23). Uma propriedade que também é preenchida para a banda é a RowCount, especificando que seu valor deve ser equivalente a "1", ou em outras palavras, que banda deve conter pelo menos uma linha, lembrando que, caso não seja definida, a banda não é exibida, uma vez que não está vinculada a um DataSet. Em seguida, é criado um objeto TfrxMemoView (linha 24), responsável pela exibição de textos, variáveis, etc, para a banda. Também são definidas algumas propriedades para o novo Memo, tais como tamanho e alinhamento (linhas 26 a 34). Uma propriedade preenchida que merece destaque é a Text, que por sua vez recebeu o valor de "[conteudo]", o nome da variável utilizada no procedimento auxiliar criado anteriormente, para que em tal Memo, seja exibido o valor presente na variável. Para concluir, o relatório (Report) tem seu evento OnGetValue (linha 37) atribuído com o procedimento RelatorioGetValue, o mesmo é preparado (PrepareReport – linha 38), exibido (ShowReport – linha 39) e liberado da memória (FreeAndNil – linha 42).

Como foi possível notar, o FastReport traz consigo outras possibilidades além da integração com tabelas do banco de dados e DataSets, onde o desenvolvedor também pode criar relatórios dinamicamente contendo o conteúdo que desejar.

Conclusão

O FastReport é uma ferramenta que reúne uma diversidade de recursos. Através dele, o desenvolvedor pode elaborar relatórios tradicionais ou dinâmicos de uma maneira muito fácil e rápida, conseguindo aumentar sua produtividade, empregando funcionalidades interessantes como é o caso de variáveis e eventos.

Esta ferramenta também é uma ótima escolha quando existe a necessidade de elaborar relatórios do tipo Cross-Tab, uma vez que possui componentes destinados a esta tarefa que automatizam e simplificam o processo.

Quanto à criação de relatórios em tempo de execução, este é um processo que raramente é empregado, porém esta ferramenta oferece todos os recursos e métodos necessários para esta finalidade, caso tal tarefa tenha que ser empregada.

Por fim, sem dúvida o FastReport é uma ferramenta que vale a pena investir, afinal é sólido e traz diversas funcionalidades que auxiliam no desenvolvimento dos mais variados projetos, contando com particularidades que muitas vezes não estão presentes em outras ferramentas.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Site do FastReport
http://www.fast-report.com

Download do FastReport para FireMonkey
http://www.fast-report.com/en/download/fast-report-firemonkey.html

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Sistema de créditos DevMedia

Conteúdo **completo** ao seu alcance

Com o sistema de créditos DevMedia você pode comprar individualmente mais de 4.000 vídeo aulas, 8.000 artigos, além de palestras e vídeos com formações completas.

TODAS as vídeo aulas que você sempre quis ver a partir de

R$ 1,20

ECONOMIA: você não precisa assinar 8 produtos diferentes para ter acesso ao conteúdo completo do site!

Saiba mais sobre o Sistema de Créditos!

www.devmedia.com.br/creditos

DevMedia

Somos tão apaixonados por tecnologia que o nome da empresa

diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,

assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única. Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486